UNIVERSITÉ DU QUÉBEC

MÉMOIRE

PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN ÉLECTRONIQUE INDUSTRIELLE

PAR

JUEGOUO JOSIANE

NOUVELLE MÉTHODE DE GÉNÉRATION DES TRAJECTOIRES POUR LA COMMANDE DE BRAS DE ROBOTS

Février 1994

À STÉPHANIE

RÉSUMÉ

Le présent travail porte sur l'activation des bras de robots et la génération des trajectoires par une méthode n'utilisant pas la cinématique inverse. Dans cette optique, nous avons défini des paramètres dynamiques tels que les variables trinaires et les caractéristiques d'incrément; ils nous ont permis d'effectuer la mise en équations conduisant à l'élaboration de l'algorithme de génération des trajectoires. L'utilisation des paramètres définis nous permet en outre d'établir à l'avance un nombre fini de commandes pouvant être appliquées à chacune des articulations. Une combinaison de commandes appliquées entraîne un déplacement élémentaire de l'organe terminal du bras. Une trajectoire est donc générée par un choix judicieux de combinaisons produisant les déplacements élémentaires successifs adéquats. Les simulations qui ont été réalisées par les algorithmes présentés nous ont permis de conclure que la méthode donne une bonne approximation des trajectoires et offre d'excellentes possibilités pour des applications telles que l'assemblage, les manipulations en milieu hostile...etc

REMERCIEMENTS

Je remercie particulièrement mon directeur de maîtrise, monsieur Aloïs KADIMA, pour tout l'enseignement que j'ai reçu de lui au cours de ces années passées à l'U.Q.T.R., et qui m'a permis d'avoir une meilleure ouverture d'esprit sur la recherche scientifique.

Je remercie également tous les professeurs du département d'ingénierie qui ont participé d'une manière ou d'une autre à ma formation.

Enfin, je tiens à remercier chaleureusement mes parents et amis pour leur soutien moral et affectif.

TABLE DES MATIÈRES

LISTE DES SYMBOLES

Les vecteurs sont désignés en caractère gras dans le texte

a_i : Paramètre D-H désignant la longueur d'un segment *i*

d_i : Paramètre D-H: distance mesurée le long de z_{i-1}

α_i : Paramètre D-H: angle mesuré entre z_{i-1} et z_i

Θ_i : Paramètre D-H: angle mesuré entre x_{i-1} et x_i

q_i : Variable articulaire pour l'articulation *i*

X : Vecteur des coordonnées spatiales

q : Vecteur des coordonnées articulaires

STEP: Pas articulaire

k : Facteur de dénormalisation du pas articulaire

c_i : Caractéristique d'incrément ou commande

del: Déplacement élémentaire

indice + : Désigne le point suivant

T : Matrice de transformation

T_{com}: Période d'échantillonage des commandes

T_x : Temps d'exécution du déplacement + retour d'information

T_A : Période d'attente

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

La recherche de solutions aux problèmes scientifiques ou technologiques par les chercheurs ne se fait pas toujours avec l'impartialité et le sens critique nécessaires. La conséquence est que très souvent, on continue pendant des années à utiliser des méthodes ou des procédés parce qu'ils donnent de bons résultats, alors qu'en les modifiant ou tout simplement en abordant le problème sous un angle différent on peut trouver des solutions plus simples, moins coûteuses et parfois plus performantes.

C'est dans cette optique que nous avons abordé le problème de la génération des trajectoires des bras de robots. En effet, la recherche bibliographique montre que toutes les méthodes de commande pour l'activation des bras articulés utilisent des algorithmes de cinématique inverse dont les perfomances ont été démontrées mais qui conservent toute leur complexité. Dans le domaine de la robotique, l'évolution rapide de la technologie des processeurs et des capteurs focalise la recherche sur des aspects particuliers tels que la vision et autres contrôles sensoriels. Aussi avons-nous voulu sortir des sentiers battus en abordant le développement d'une nouvelle méthode de génération des trajectoires.

Les objectifs de ce travail sont les suivants:

- concevoir des algorithmes simples permettant de contourner la complexité des algorithmes de cinématique inverse;
- éviter l'utilisation de calculateurs puissants et coûteux;
- améliorer le temps de calcul.

Pour réaliser nos objectifs, nous nous sommes donné comme méthodologie la mise en équation générale des robots à partir de la cinématique directe, la formulation des paramètres dynamiques de notre méthode, l'élaboration des algorithmes et enfin l'écriture des programmes et la simulation.

Pour l'écriture des programmes, la méthode a été appliquée à une structure particulière; en l'occurence, un bras articulé à six axes de rotation. Cette restriction ne nuit pas à la généralité du principe de la méthode. Nous admettons comme hypothèse de départ que le niveau de technologie actuel permet de construire des actionneurs capables de répondre à nos besoins.

CHAPITRE I

PRINCIPES DE BASE DE LA ROBOTIQUE

1.1 Généralités

Au cours des siècles, le désir de l'homme de créer des outils de travail de plus en plus perfectionnés, puissants et efficaces l'a conduit à la création des machines. C'est en voulant donner à ces machines une certaine autonomie qu'est apparue l'automatisation. L'homme s'est alors rendu compte qu'en construisant des machines ayant une structure appropriée, elles pouvaient effectuer plusieurs tâches différentes au lieu d'une seule. Ainsi est née la première vague de robots industriels. Cependant, depuis la commercialisation du premier robot industriel par la firme UNIMATION en 1955, la robotique a connu de nombreux progrès; elle suit d'ailleurs l'évolution des technologies nouvelles telles que l'intelligence artificielle et la miniaturisation des puces.

Néanmoins, malgré le fait qu'on s'achemine de nos jours vers des robots de plus en plus adaptatifs, capables de décision et aux structures géométriques plus ou moins variées, les fondements de la robotique restent les mêmes. Seules viennent s'y

greffer des techniques de construction, de contrôle et de commande de plus en plus sophistiquées.

1.2 Caractéristiques principales d'un robot

1.2.1 Le robot dans son contexte

On ne peut donner une définition précise du robot, car il n'existe pas de robot universel apte à accomplir n'importe quelle tâche humaine. Un robot m'apparaît comme tel que dans un environnement donné et par rapport à ce qui l'entoure; il est conçu pour effectuer une famille de tâches dans un type de milieu bien déterminé. Cependant, on retrouve toujours autour du robot, trois composantes essentielles. (fig. 1.1).

- **La source d'énergie** : elle peut être électrique, hydraulique ou pneumatique;

- **La source d'information** : elle définit les tâches à accomplir et traite toutes les informations relatives aux commandes et aux mesures;

- **L'environnement** : il comprend à la fois l'espace atteignable du robot, les obstacles, les objets à manipuler et les propriétés physiques du milieu.

Le robot, quant à lui, est le système mécanique dont la structure géométrique varie suivant le type des tâches pour lesquelles il a été conçu. Il est doté d'actionneurs alimentés par la source d'énergie, d'un organe effecteur

qui interagit avec l'environnement, et de capteurs qui informent l'organe de traitement de l'information de l'état actuel du robot.

On peut donc schématiser un robot fonctionnel tel que l'indique la figure 1.2.

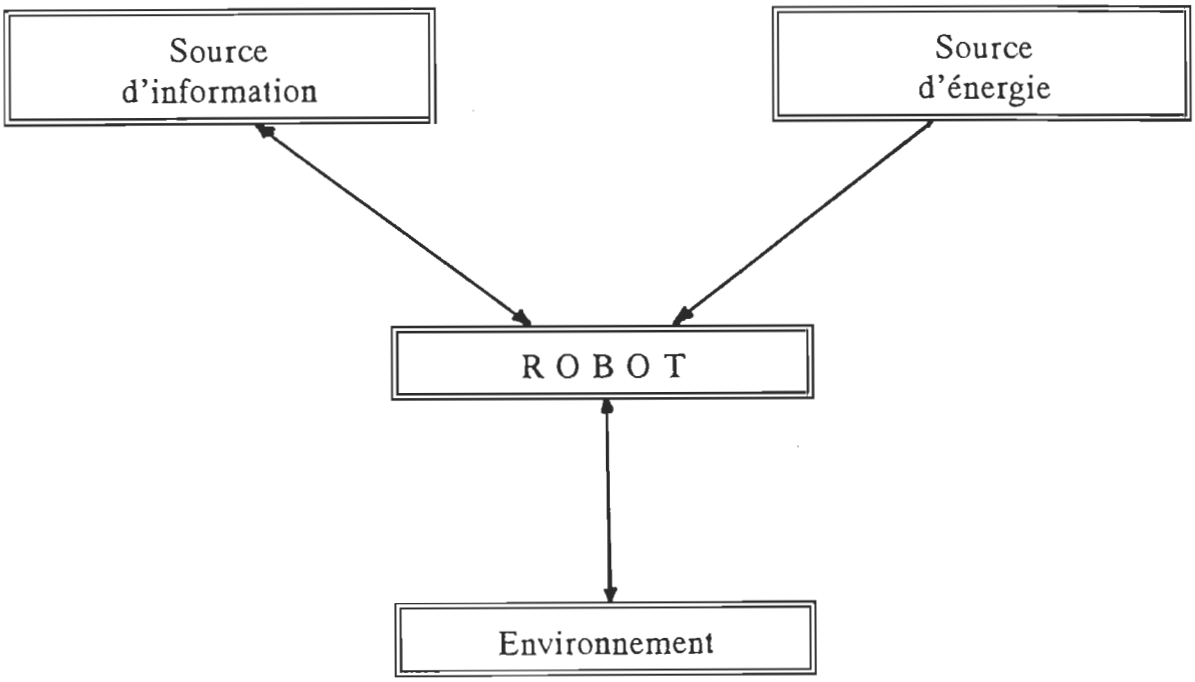


Figure 1.1 : Le robot dans son contexte

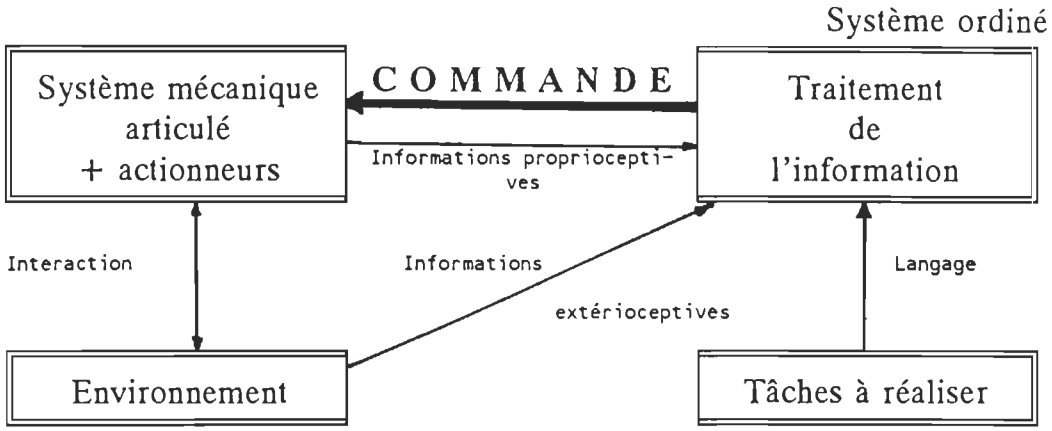


Figure 1.2 : Composantes d'un robot en fonctionnement

1.2.2 Les propriétés du robot

Le passage de la machine automatique au robot, résulte du besoin de rendre la machine apte à remplacer un opérateur humain - autant que faire se peut - dans l'exécution de certains types de tâches. Il en découle deux propriétés essentielles dont les constructeurs essaient de munir au mieux les robots : ce sont la versatilité et l'adaptativité.

1.2.2.1 La versatilité

C'est la propriété rattachée à la structure géométrique et mécanique du robot. Elle définit son aptitude à atteindre des positions et des orientations physiques diverses et par ce fait, sa potentia-

lité à exécuter une multitude de tâches d'une part et, d'autre part, sa potentialité à exécuter la même tâche de plusieurs façons différentes. La versatilité caractérise donc un robot à géométrie variable. Elle ne dépend pas uniquement du nombre de degrés de liberté, mais aussi d'autres paramètres tels que la structure du poignet par exemple.

1.2.2.2 L'adaptativité

Le niveau de technologie actuel permet de parler plutôt d'auto-adaptativité. Cette propriété caractérise la capacité du robot à prendre des initiatives afin d'exécuter correctement une tâche partiellement décrite ou non entièrement spécifiée, et ceci même en présence de modifications imprévues de l'environnement. Cette propriété que l'on s'efforce de perfectionner, est en général limitée par les performances des capteurs, la vitesse des calculateurs et les imperfections des algorithmes d'intelligence artificielle.

1.3 Structure générale d'un robot

L'architecture géométrique du robot varie selon le type des tâches qui lui sont destinées. L'un des éléments qui caractérise la géométrie du robot est le nombre de degrés de liberté.

1.3.1 Notion de degré de liberté

Dans l'espace réel à trois dimensions, un solide indéformable libre possède trois possibilités de translation et trois possibilités de rotation. On dit alors que le solide a six degrés de liberté; sa position, à un instant donné, peut être décrite par six paramètres indépendants : trois paramètres de rotation définissant l'orientation du solide par rapport à un trièdre fixe R_o, et trois paramètres décrivant les coordonnées d'un point particulier du solide dans ce même trièdre. Ainsi, les possibilités de translation assurent le déplacement tandis que les possibilités de rotation assurent l'orientation du solide.

1.3.2 Éléments principaux et degrés de liberté du robot

D'une manière générale, on distingue au robot trois parties essentielles dont la mobilité dans l'espace va définir les degrés de liberté du robot. Il s'agit :

- du véhicule;
- du porteur;
- de l'organe terminal.

1.3.2.1 Le véhicule

Son rôle est d'amener le robot dans la partie de son environnement où il a une tâche à accomplir. Néanmoins, le véhicule n'est pas toujours existant. Dans le cas le plus général où le robot

est, par exemple, lié à un satellite évoluant dans l'espace, le véhicule possède trois degrés de liberté en translation et trois degrés de liberté en rotation. Cependant, dans un environnement terrestre, on supprime habituellement au véhicule les trois degrés de liberté en rotation pour ne lui laisser que ceux en translation (ou deux des trois) afin d'assurer son déplacement. Pour les robots fixes, on dira que le véhicule - qui est la base du robot - n'a aucun degré de liberté.

1.3.2.2 Le porteur

Le porteur est la structure mécanique, généralement une structure articulée, dont le rôle est d'amener l'extrémité du robot en divers points de son espace de travail. D'une manière générale le porteur possède les trois degrés de liberté en translation qui assurent le déplacement; cependant, il peut en posséder moins, dépendamment du type de tâches assignées au robot.

1.3.2.3 L'organe terminal

C'est sur l'organe terminal qu'est fixé l'outil. Une fois que ce dernier a été positionné grâce au porteur, il faut lui donner l'orientation adéquate pour l'exécution de la tâche. C'est donc l'organe terminal qui assure l'orientation de l'outil. Pour avoir une bonne précision d'orientation, il faut fournir à l'organe terminal trois

degrés de liberté en rotation; l'orientation est d'autant meilleure que l'on peut effectuer trois rotations autour de trois axes normaux et concourants.

Ainsi, un robot permettant d'effectuer un mouvement absolument quelconque (tel un bras manipulateur lié à un satellite évoluant dans l'espace) devrait posséder 12 degrés de liberté ou au moins 9 (si on ôte au véhicule ses possibilités de rotation). Cependant, la plupart des robots usuels ont un "véhicule fixe" et se limitent ainsi à six degrés de liberté et moins.

1.3.3 Nombre de degrés de liberté du robot

La plupart des robots usuels ayant n degrés de liberté ($n \leq 6$) possèdent également n axes, qu'ils soient de translation ou de rotation. Cependant, le nombre d'axes du robot ne définit pas nécessairement le nombre de degrés de liberté.

Si on considère le robot fixe, sans véhicule, les six degrés de liberté possibles sont répartis entre le porteur et l'organe terminal de manière que les trois rotations soient liées à ce dernier. Pour compenser un degré de liberté déficitaire au niveau de l'organe terminal, on rajoute en général un ou plu-

sieurs axes au niveau du porteur. On obtient ainsi des robots pouvant avoir sept axes et plus, mais avec six degrés de liberté (ex.: robot peintre ACMA).

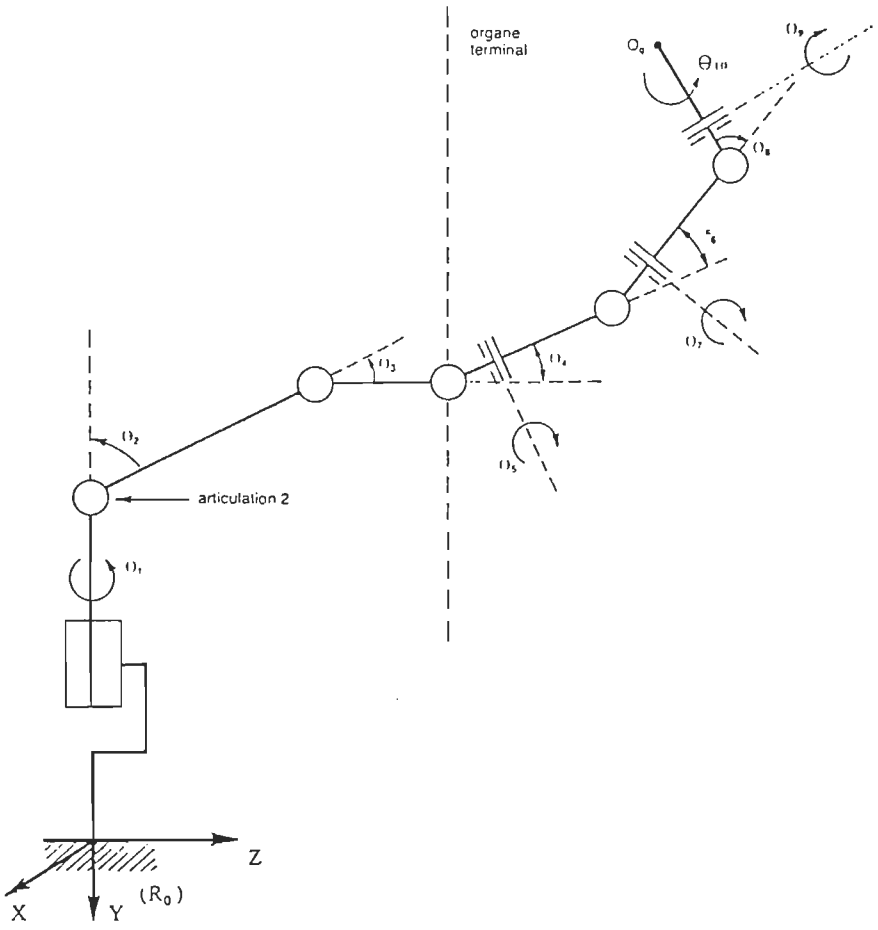


Figure 1.3 : Structure du robot peintre ACMA
(10 axes, 6 degrés de liberté)[1]

[1] Philippe COIFFET. Les robots. Modélisation et commande. Tome 1. Hermes Publishing (France). 1981. p. 22.

1.4 Morphologie des porteurs

La structure d'un porteur est caractérisée par des corps rigides, les segments, susceptibles de se mouvoir par rapport à une base et par des articulations limitant le mouvement relatif entre deux segments adjacents.

Si les segments peuvent avoir des formes variées, les articulations qui sont des liaisons bilatérales ont par contre des caractéristiques bien précises. On peut ainsi distinguer deux groupes d'articulations:

a) Les articulations simples

Elles sont caractérisées par une mobilité unique : c'est-à-dire que la liaison entre deux segments adjacents permet un seul mouvement relatif. Il s'agit :

- **des articulations rotoïdes** : leur liaison est de type charnière et permet une rotation autour d'un axe commun aux deux segments adjacents;

- **des articulations prismatiques** : leur liaison est de type glissière et permet une translation le long d'un axe commun aux deux segments adjacents.

b) Les articulations complexes

Elles autorisent plusieurs degrés de mobilité et sont en général constituées par des dispositifs composés eux-mêmes de plusieurs articulations simples.

Cependant, en robotique, chaque articulation est généralement motorisée. De ce fait, les porteurs sont presque toujours composés exclusivement d'articulations simples.

1.5 Classification des porteurs

Les porteurs sont souvent classés, du point de vue de la structure, suivant le système de coordonnées dans lequel ils opèrent. On définit ainsi quatre classes principales pour les robots :

- les robots cartésiens;
- les robots cylindriques;
- les robots sphériques;
- les robots articulés.

1.5.1 Les robots cartésiens

Les porteurs cartésiens possèdent trois degrés de liberté en translation. Toutes les articulations sont de type glissère (fig. 1.4)

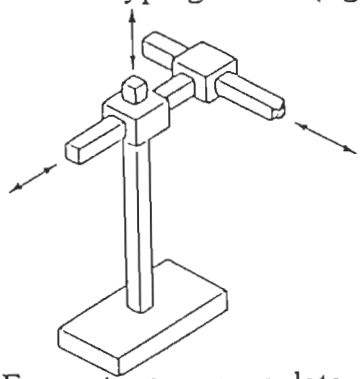


Figure 1.4 : Exemple de manipulateur cartésien[2]

[2] Ho, C.Y. and SRIWATTANATHAMMA, Jen. ROBOTS KINEMATICS: symbolic automation and numerical synthesis. A.P.C. 1990.

1.5.3 Les robots sphériques

Les porteurs travaillant en coordonnées sphériques ont une structure de base comportant deux rotations et une translation; il s'agit alors d'un bras téléscopique pouvant effectuer une rotation autour d'un axe vertical et une rotation autour d'un axe horizontal, ce qui donne à l'espace de travail du porteur la forme d'une coquille sphérique. Comme pour les porteurs cylindriques, l'existence des rotations réduit considérablement la résolution en bout de bras à cause de l'amplification des erreurs angulaires apparaissant au niveau des axes de rotation. Néanmoins, l'avantage d'une telle structure est qu'elle améliore la flexibilité dans l'utilisation.

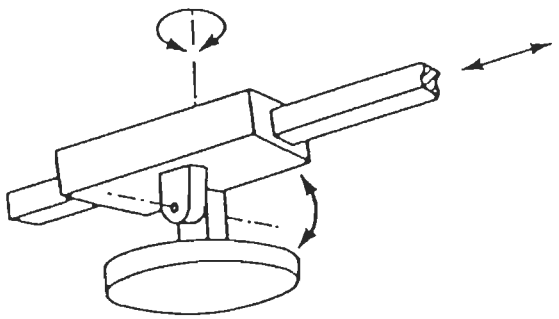


Figure 1.6 : Exemple de manipulation sphérique[2]

1.5.4 Les robots articulés

Les robots articulés ont une structure qui rappelle la plupart du temps celle d'un bras humain. En général, ils comprennent quatre segments principaux que l'on peut assimiler au tronc, au bras, à l'avant-bras et à la main; toutes les articulations du porteur articulé sont donc de type rotoïde et vont souvent correspondre à l'épaule, au coude et au poignet. De ce fait, la résolution en bout de bras du porteur articulé, quoiqu'elle dépende de la position de travail, est toujours très mauvaise. Cependant, de toutes les structures, la structure articulée est celle qui offre la meilleure flexibilité et permet une grande souplesse dans l'exécution des tâches, d'où son utilisation fréquente dans les systèmes robotisés de petite et moyenne envergure. De plus, grâce aux progrès technologiques en matière d'asservissement, les robots articulés peuvent être utilisés pour des applications nécessitant une très grande précision.

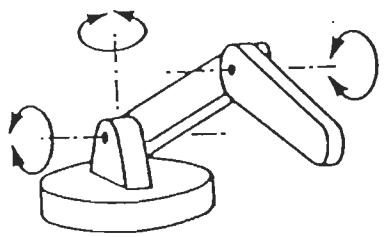


Figure 1.7 : Exemple de manipulateur articulé[2]

1.6 Commande dynamique des servomécanismes

1.6.1 Généralités

L'animation du porteur consiste à mouvoir ses articulations, quelle que soit leur structure (rotoïde ou prismatique). Pour ce faire, on utilise des actionneurs qui convertissent une énergie primaire en énergie mécanique permettant de produire le mouvement de rotation ou de translation nécessaire. En robotique, les asservissements permettant l'activation des différentes articulations sont réalisés à l'aide d'actionneurs qui peuvent être électriques, hydrauliques ou pneumatiques, suivant l'importance du système robotisé et la nature des tâches à accomplir. On les appelle des servomécanismes parce que leur grandeur de sortie est une fonction mécanique. Ils sont caractérisés par la présence d'une ou plusieurs boucles de contre-réaction qui permettent une action en fonction de la comparaison entre la valeur de consigne et celle que fournit l'organe de mesure.

1.6.2 Les actionneurs hydrauliques et pneumatiques

Ils mettent en oeuvre l'énergie fluidique qui est véhiculée par un fluide liquide ou gazeux, mis sous pression à l'aide d'une pompe ou d'un compresseur et circulant dans des canalisations.

Dans la plupart des applications de robotique, la transformation de l'énergie fluidique en énergie mécanique se fait par l'exploitation directe de

l'énergie potentielle du fluide. Les écoulements se font donc à basse vitesse, ce qui est propice aux fonctionnements intermittents qui caractérisent les robots; par ailleurs cela permet de développer des efforts à l'arrêt lorsque cela est nécessaire. On obtient une force ou un couple de poussée du fluide agissant sur les éléments mobiles de l'actionneur proportionnellement à une différence de pression :

$f = S\ (p_1 - p_2)$ f est la force, S est une surface

$c = C\ (p_1 - p_2)$ c est le couple, C est un volume

* **La génération de l'énergie fluidique est réalisée à l'aide :**

- d'une source d'énergie mécanique (moteur thermique ou électrique)
- d'un convertisseur d'énergie mécanique en énergie fluidique (pompe ou compresseur ...)
- d'un régulateur de pression
- d'un réservoir de stockage du fluide

* **L'utilisation de l'énergie fluidique nécessite :**

- une canalisation aller, à haute pression
- une canalisation retour, à basse pression
- un modulateur agissant sur le débit et/ou sur la pression du fluide
- un actionneur qui convertit l'énergie véhiculée par le fluide en énergie mécanique

- des organes auxilliaires : filtres, clapets des surpressions, clapets anti-retour.

1.6.2.1 Commande des actionneurs pneumatiques

Les organes principaux des actionneurs pneumatiques sont généralement commandés en "tout-ou-rien" par des distributeurs à clapet ou des distributeurs à tiroir. La commutation des distributeurs est réalisée par le déplacement de pièces mobiles provoqué par l'utilisation de relais à membranes ou d'électro-aimants à noyau plongeur ou à armure rotative.

Les asservissements de position des actionneurs pneumatiques utilisent essentiellement des techniques proportionnelles; cependant, la nature du fluide et les frottements secs entre les pièces mobiles ne permettent pas de réaliser des distributeurs progressifs ou des servovalves de précision.

Néanmoins, pour améliorer la précision en positionnement des actionneurs pneumatiques, on s'efforce de réaliser des distributeurs de faible inertie à deux positions (tout-ou-rien) ou à trois positions (plus-ou-moins). Leur commande étant faite par le signe de l'écart de position, ou par la modulation de la durée ou de la fré-

quence d'implusion de pilotage. L'utilisation d'un micro-processeur permet alors de réaliser un meilleur asservissement.

1.6.2.2 Commande des actionneurs hydrauliques

Les systèmes robotisés de grosse puissance utilisent essentiellement des actionneurs hydrauliques dont les principaux problèmes sont liés aux fuites d'huile (étanchéité), à la filtration et à la purge de l'air parfois retenu dans les canalisations, et ce, quel que soit le type d'actionneur.

Ces actionneurs sont généralement commandés par des servovalves dont la fonction est de déplacer le tiroir d'un distributeur de façon à libérer des sections de passage de fluide proportionnelles à un signal électrique de commande qui peut être une tension ou un courant. Cependant, à cause du moteur-couple, la servovalve est entachée d'hystérésis, ce qui rend difficile l'obtention simultanée d'un débit élevé permettant une grande vitesse de l'actionneur et d'une très bonne précision de positionnement.

Pour concilier ces deux caractéristiques et améliorer les performances dynamiques des servovalves, on réalise de plus en plus

un asservissement électrique direct de la servovalve avec un moteur-couple de puissance élevée, attaqué par une électronique appropriée.

1.6.3 Les actionneurs électriques

Une très large gamme d'applications en robotique relève du domaine des petites et moyennes puissances. Pour ces applications, on utilise surtout des actionneurs électriques à cause de leurs nombreux avantages dont quelques-uns sont :

- le transport facile de l'énergie;
- la commande précise, homogène, aisée et fiable;
- l'absence de fuite et de pollution;
- la possibilité de rendre le robot autonome en utilisant des batteries;
- La possibilité de monter les actionneurs directement sur les axes.

Leur principal inconvénient est en général leur poids.

Les actionneurs électriques les plus utilisés pour la motorisation des robots sont :

- les moteurs pas à pas;
- les moteurs à courant continu.

1.6.3.1 Commande des moteurs pas à pas

Les moteurs pas à pas ont l'avantage de permettre la conversion directe d'un signal électrique digital en un positionnement angulaire de caractère incrémental. Cependant la principale difficulté dans son utilisation est l'instabilité liée à l'accélération et la décélération.

L'intérêt principal dans l'utilisation du moteur pas à pas est sa possibilité de fonctionner en boucle ouverte : un certain nombre d'impulsions assure un décalage angulaire bien déterminé.

Néanmoins, selon le type de moteur et l'usage qu'on en fait, une électronique plus élaborée peut lui être associée. Cependant, la commande des moteurs pas à pas exige que l'on puisse contrôler parfaitement le courant, car leur couple dépend presque exclusivement du courant.

La commande en boucle ouverte des moteurs pas à pas doit satisfaire à certains critères :

- **le passage** d'une position à une autre doit se faire en temps minimum;
- **le fonctionnement** doit rester synchrone pour qu'il n'y ait pas de perte de pas;

- **le point final** doit être atteint sans oscillation.

Lorsque la charge est variable, il faut prévoir une commande adaptative, car la charge intervient directement dans la réponse du système.

La commande est plus aisée si la gestion des informations est faite à l'aide d'un microprocesseur (fig. 1.8).

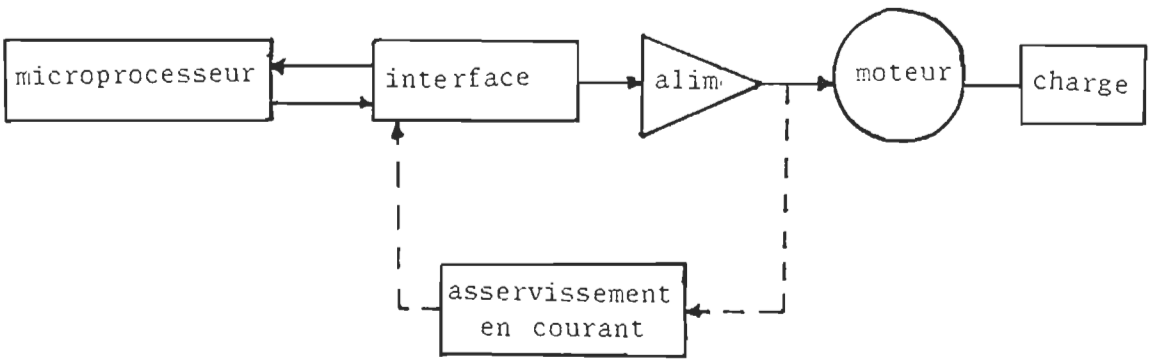


Figure 1.8 : Schéma théorique de commande d'un moteur pas à pas

Par ailleurs la technologie actuelle permet de réaliser des circuits intégrés pour la commande des moteurs. La figure 1.9 nous montre un exemple de circuit tiré d'un catalogue de International Rectifier: il s'agit du IR8200 B.

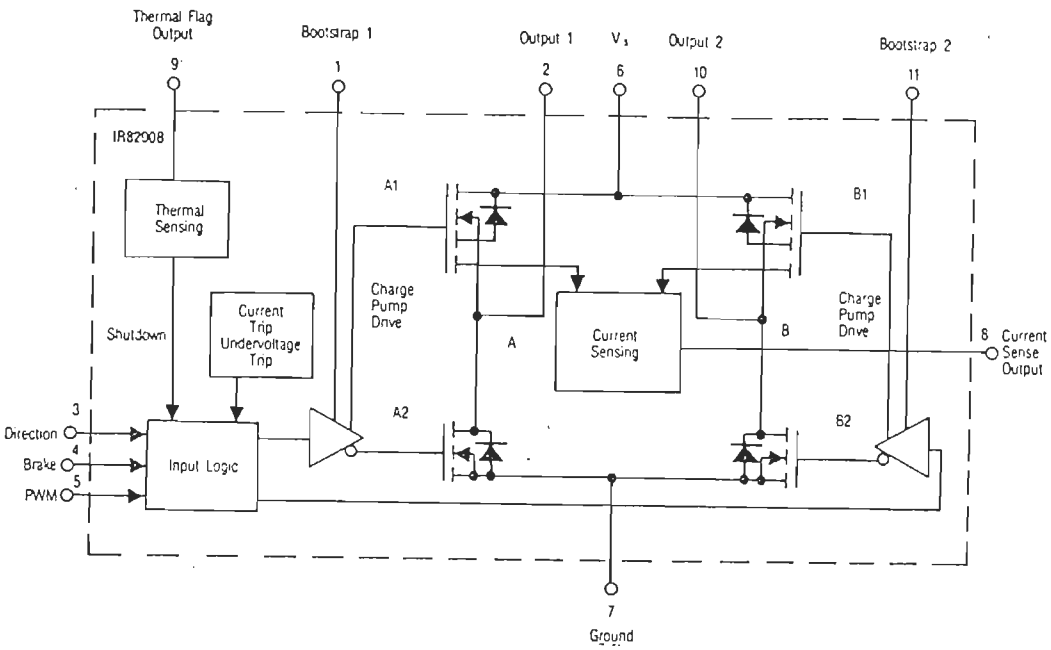


Figure 1.9 : Exemple de circuit de commande pour un moteur pas à pas

1.6.3.2 Les moteurs à courant continu

Les différents types de moteurs à courant continu souvent utilisés en robotique sont :

- **les moteurs à induit bobiné classique** qui sont robustes et par leur construction ont une forte capacité thermique;

- **les moteurs à induit discoïdal**; ils ont une constante de temps mécanique réduite; cependant ils présentent deux constantes de temps thermiques dont celle de l'induit est relativement faible;

- **les moteurs à induit en cloche**; ils ne peuvent servir efficacement que dans le cas où le couple nécessaire en régime permanent est faible;

- **les moteurs toroïdaux**; ils permettent d'obtenir soit des couples très élevés avec des vitesses de rotations faibles, soit des vitesses très élevées pour des couples très faibles.

Les moteurs à courant continu sont des actionneurs dont l'utilisation est très souple; ils possèdent un réglage facile de la vitesse et de la position et un contrôle aisé du couple. Ils ne posent pas de problème de stabilité intrinsèque et ont un domaine de fonctionnement à priori illimité. Néanmoins, des limites leur sont imposées par des phénomènes annexes tels que :

- l'échauffement du moteur;
- la démagnétisation;
- la commutation;
- les contraintes de tenue mécanique des enroulements et autres pièces rotatives.

1.6.3.3 Commandes des moteurs à courant continu

Dans l'étude de l'asservissement des servomécanismes, il est aisé d'utiliser comme modèle de base un modèle mathématique linéaire. Dans le cas des moteurs à courant continu, le modèle linéaire est obtenu si on admet au préalable les hypothèses simplificatrices suivantes :

- le moteur est compensé : l'effet de réaction d'induit est par conséquent nul;
- le frottement est purement visqueux : on annule donc le frottement sec;
- les jeux sont nuls et l'arbre est rigide;
- le moteur fonctionne en zone non saturée.

La mise en équation consiste à appliquer au système les lois physiques qui le régissent. Si on considère que le servomécanisme suivant commande une articulation d'un porteur articulé, on peut en établir le schéma fonctionnel de régulation; cela permet, en faisant l'analyse temporelle, d'effectuer la simulation d'un asservissement en position.

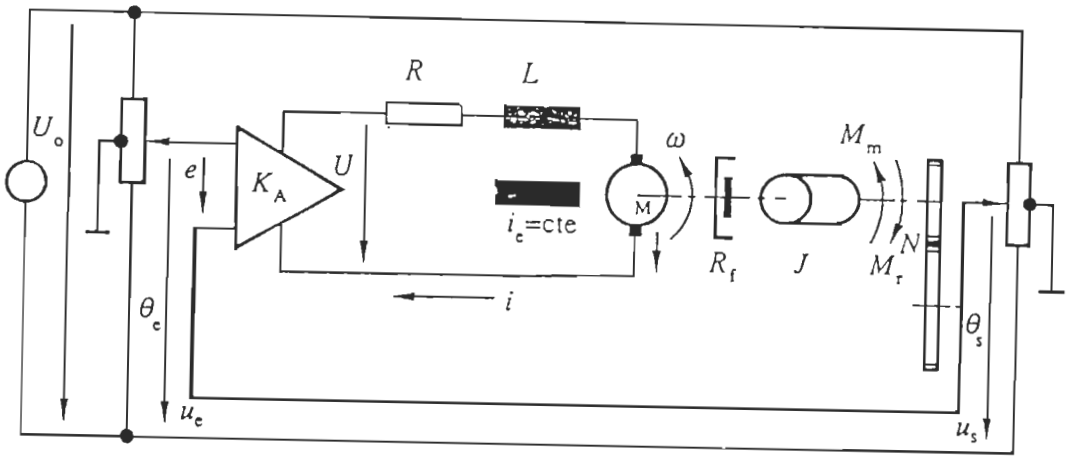


Figure 1.10 : Schéma d'un servomécanisme : régulation de position

L'étude des réponses temporelles d'un système dynamique à diverses excitations permet de déterminer ses principales performances, à savoir :

- la stabilité que l'on évalue par la marge de gain;
- la précision statique et la précision dynamique que l'on évalue par la valeur des signaux d'erreurs permanents;
- la rapidité qui est donnée par le temps de réponse;
- la qualité de la régulation.

La condition d'avoir une bonne stabilité introduit généralement dans les systèmes à régulation automatique un dilemme stabilité-précision qui n'est résolu que par l'introduction de régulateurs, très souvent sous forme de correcteurs.

En robotique, en plus de réaliser des asservissements stables et précis, il est important d'utiliser des systèmes à réponse rapide car plus la constante de temps mécanique est faible, plus la réponse dynamique du système est rapide;ce qui permet de réduire la période d'échantillonnage des processeurs (pour l'envoi des signaux de commande).

En effet, si on écrit les équations électromécaniques qui régissent le régime transitoire d'un moteur à courant continu, la relation mécanique traduit le fait qu'au couple résistant C_r s'ajoute le couple dû à l'inertie des pièces mises en accélération, soit :

$$C_m = C_r + J \frac{d\omega}{dt} \qquad (1.1)$$

où C_m est le couple moteur durant le régime transitoire et J est la somme de J_o, inertie propre du moteur et J_m inertie de la charge ramenée au moteur.

La relation électrique met en exergue la loi qui existe entre le courant *i* et la tension U appliquée au moteur, soit :

$$U = Ri + L \frac{di}{dt} + e \qquad (1.2)$$

où

- e est la force contre-électromotrice du moteur (fcem)

- R et L sont respectivement la résistance et l'inductance du moteur

Si on suppose le moteur à l'arrêt et qu'on applique brusquement à ses bornes un échelon de tension U_o avec comme hypothèse que :

- le moteur n'est pas chargé et les frottements sont nuls, donc $C_r = 0$ et $J_m = 0$;
- l'inductance L est négligeable : $L = 0$;
- les pertes par effet Joule sont négligeables.

Des équations (1.1) et (1.2) on obtient :

$$C_e = J_o \frac{d\omega}{dt} = k_m\, i$$

(k_m = constante de couple ou constante de fcem)

$$U_o = Ri + k_m\omega \qquad (e = k_e.\omega = k_m.\omega$$

on en déduit que : $$\frac{d\omega}{dt} = k_m \frac{U_o - k_m\omega}{R.J_o} \qquad (1.3)$$

Lorsqu'on atteint la vitesse d'équilibre ω_0 , on a simultanément :

$$\frac{d\omega}{dt} = 0 \quad et \quad i = 0 \quad \Rightarrow \quad \omega_o = \frac{U_o}{k_m}$$

on peut donc réécrire (1.3) :

$$\frac{d\omega}{dt} = \frac{k_m^2}{J_o . R}(\omega_o - \omega) \tag{1.4}$$

L'équation (1.4) est une équation différentielle dont la solution pour la condition initiale $\omega = 0$ à $t = 0$ est:

$$\omega = \omega_o(1 - e^{-t/\tau_m}) \tag{1.5}$$

dans laquelle $\tau_m = \frac{J_o . R}{k_m^2}$

τ_m caractérise la montée en vitesse et est appelée "constante de temps mécanique". D'après l'équation (1.5) la montée en vitesse se fait selon une loi exponentielle, aussi sera-t-elle d'autant plus rapide que la valeur de τ_m est faible.

CHAPITRE II

RÉSOLVABILITÉ CLASSIQUE DES ROBOTS

2.1 Matrices de transformation homogène

Le problème cinématique fondamental en robotique est la formulation des équations permettant de transformer l'espace des coordonnées articulaires en espace cartésien. En effet, la motorisation du robot et par conséquent la commande se fait au niveau des articulations, tandis que la tâche à exécuter, et par ce fait, la position et l'orientation de l'organe effecteur se déroule dans l'espace cartésien.

Ceci explique la nécessité de définir un outil mathématique permettant d'analyser un système robotique quelle que soit sa configuration géométrique. La plupart des ouvrages d'introduction à la robotique décrivent plus en détail les notions mathématiques permettant d'aboutir à l'écriture des matrices de transformation.

Rappelons néanmoins que l'on peut décrire un bras de robot comme étant une chaîne cinématique de segments rigides. Deux segments adjacents sont liés par une articulation ayant un seul degré de liberté en rotation ou en translation. Chaque segment est entièrement décrit par sa position et son orientation.

2.1.1 Position et orientation d'un corps rigide

Considérons un repère fixe R_o, d'axes x_o, y_o, z_o, et un corps rigide dans l'espace représenté par un point O_p. La position du point O_p est décrite dans le repère fixe par le vecteur colonne 3 x 1 :

$$p = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{2.1}$$

Si on associe au corps rigide un repère orthonormé d'axes x_p, y_p, z_p centré en O_p (fig. 2.1), alors, ce repère qui bouge avec le corps permet de définir son orientation par rapport au repère fixe R_o. En effet, soient **n, o** et **a**, les vecteurs unitaires liés respectivement aux axes x_p, y_p, z_p; les composantes de chacun de ces vecteurs sont les cosinus directeurs obtenus en les projetant dans le repère fixe R_o dont les vecteurs unitaires sont **i, j** et **k**.

La rotation du corps rigide est ainsi décrite par les neuf cosinus directeurs qui forment la matrice 3 X 3 de rotation R :

$$R = \begin{bmatrix} n.i & o.i & a.i \\ n.j & o.j & a.j \\ n.k & o.k & a.k \end{bmatrix} \tag{2.2}$$

(n.i, n.j, n.k sont les cosinus directeurs de n projeté dans R_o)

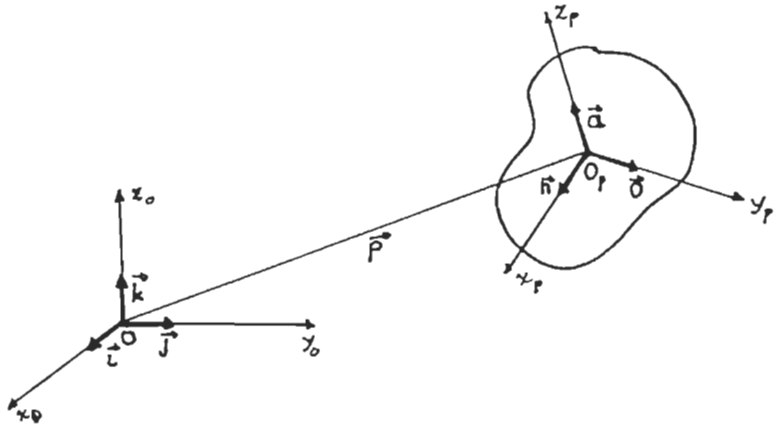


Figure 2.1 : position et orientation du corps rigide

2.1.2 Transformation homogène

Si on considère un point quelconque M de l'espace (fig. 2.2), le vecteur **OM** est défini par :

$$\mathbf{OM} = \mathbf{p} + \mathrm{R}\,\mathbf{O_pM} \tag{2.3}$$

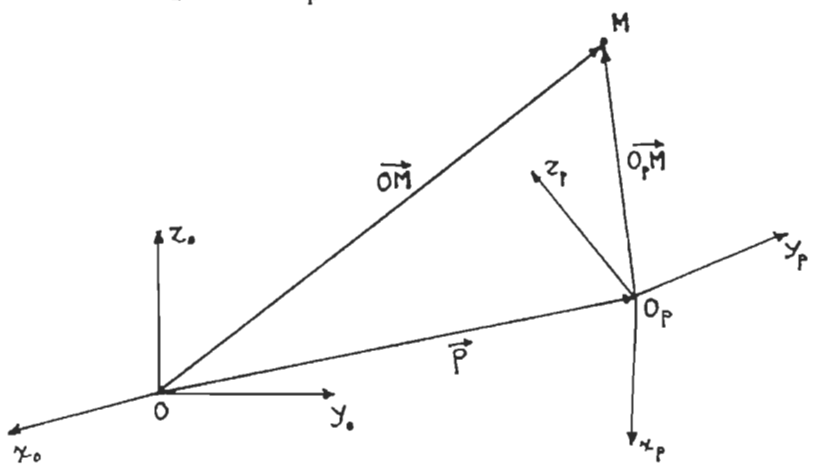


Fig. 2.2 : Transformation de coordonnées

Notons x, y, et z les coordonnées de **OM** dans $R_o$et u, v, et w les coordonnées de $\mathbf{O_pM}$ dans R_p. On peut alors écrire que :

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{bmatrix} R & \boldsymbol{p} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} \tag{2.4}$$

avec $A = \begin{bmatrix} R & \boldsymbol{p} \\ 0 & 1 \end{bmatrix}$

La matrice A représente à la fois la position et l'orientation du repère O_p-$x_p y_p z_p$.

L'équation : $\mathbf{OM} = A\ \mathbf{O_pM}$ constitue la transformation homogène permettant de passer du repère R_p au repère R_o; la matrice A est la matrice de transformation.

Notons A_o^1 la matrice de transformation qui permet de passer du repère R_1 au repère R_o; pour *n* transformations consécutives, il est possible de passer du repère R_n au repère R_o grâce à l'équation :

$$\mathbf{X}^\circ = A_o^1\ A_1^2\ \ldots\ A_{n-1}^n\ \mathbf{X}^n \tag{2.5}$$

où

$\mathbf{X}^n$ est le vecteur décrivant un point M dans le repère R_n et

$\mathbf{X}^\circ$ le vecteur décrivant ce même point dans le repère R_o.

Nous pouvons appliquer ces transformations à un bras manipulateur considéré comme une chaîne articulée de segments rigides, à chaque segment d'indice *i* étant lié un repère R_i.

Les matrices de transformation A_{i-1}^{i} permettent alors de décrire la position et l'orientation du segment *i* par rapport au segment *i-1*.

2.2 Convention de Denavit-Hartenberg

La plupart des robots industriels ou des bras manipulateurs usuels ont une structure assimilable à une chaîne cinématique ouverte, ayant en général une base fixe. Il est donc aisé de décrire pour ces robots la position et l'orientation de l'organe effecteur grâce à une succession de transformations homogènes.

Pour cela, Denavit et Hartenberg ont développé en 1955 une approche mathématique basée sur la matrice de transformation homogène et permettant de décrire de manière systématique les relations cinématiques entre les articulations.

Ces relations utilisent un nombre minimum de paramètres, soient deux distances (a_i et d_i) et deux angles (α_i et Θ_i) pour chaque segment.

Dans la convention de Denavit-Hartenberg (D-H), l'indice *i* correspond au repère R_i lié au *i-ème* segment du robot, l'axe z du repère R_i étant aligné avec l'axe de l'articulation *i+1*. (fig. 2.3)

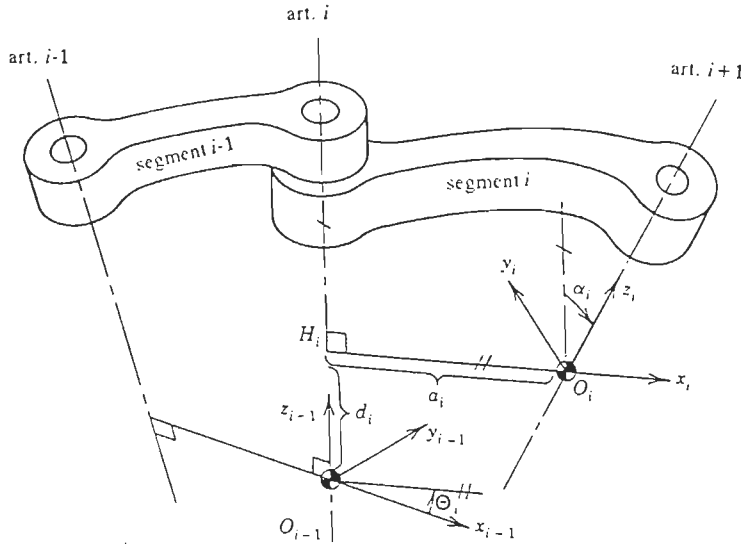


Figure 2.3 : Notation de Denavit-Hatenberg
pour une paire de segments adjacents[1]

Les quatre paramètres de la notation D-H sont décrits de la manière suivante :

a_i : il décrit la distance entre les axes z_{i-1} et z_i, mesurée le long de l'axe x_i et correspondant à la longueur du segment *i*;

d_i : c'est la distance entre l'origine O_{i-1} et l'intersection de l'axe z_{i-1} avec le prolongement de l'axe x_i; elle est mesurée le long de z_{i-1};

α_i : c'est l'angle que fait l'axe z_{i-1} avec l'axe z_i, mesuré suivant la règle de la main droite;

Θ_i : c'est l'angle entre l'axe x_{i-1} et x_i, mesuré par rapport à z_{i-1} selon la règle de la main droite.

---

[1] Ho, C.Y. and SRIWATTANATHAMMA, Jen. ROBOTS KINEMATICS: symbolic automation and numerical synthesis. A.P.C. 1990.

Pour les articulations rotoïdes, les paramètres a_i, d_i et α_i sont constants, tandis que pour les articulations prismatiques, ce sont a_i, α_i et Θ_i qui sont constants. La convention de Denavit-Hartenberg rend plus aisée la construction des matrices de transformations.

On démontre facilement que la matrice A_{i-1}^{i} qui permet de décrire le segment *i* par rapport au segment *i-1* s'écrit, pour une articulation rotoïde :

$$A_{i-1}^{i} = \begin{bmatrix} \cos\Theta_i & -\sin\Theta_i\cos\alpha_i & \sin\Theta_i\sin\alpha_i & a_i\cos\Theta_i \\ \sin\Theta_i & \cos\Theta_i\cos\alpha_i & -\cos\Theta_i\sin\alpha_i & a_i\sin\Theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.6)$$

et pour une articulation prismatique :

$$A_{i-1}^{i} = \begin{bmatrix} \cos\Theta_i & -\sin\Theta_i\cos\alpha_i & \sin\Theta_i\sin\alpha_i & 0 \\ \sin\Theta_i & \cos\Theta_i\cos\alpha_i & -\cos\Theta_i\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.6)\text{ bis}$$

Ainsi, la matrice de transformation globale qui aide à décrire la position et l'orientation du dernier segment par rapport à la base s'écrit :

$$T = A_o^1(q_1)\, A_1^2(q_2) \ldots A_{n-1}^n(q_n) \qquad (2.7)$$

où q_i est le paramètre variable de l'articulation *i*; q_1, q_2 ... q_n constituent les variables articulaires.

Lorsqu'on établit au préalable le tableau des paramètres (fig. 2.4), les matrices de transformation A_{i-1}^i sont calculées rapidement; la matrice globale T permettant d'écrire l'équation cinématique du robot est de la forme suivante :

$$T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Le vecteur $\boldsymbol{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$ décrit la position de l'organe terminal dans l'espace carté-

sien, tandis que les composantes des vecteurs *n, o* et *a* sont les cosinus directeurs qui permettent de décrire l'orientation de l'organe terminal; ils constituent la matrice d'orientation R.

$$R = \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix}$$

Paramètres N° d'articulation	a_i	α_i	d_i	Θ_i
1				
2				
...	...	...	...	...
n				

Figure 2.4. : Tableau des paramètres D-H

2.3 Résolution du problème cinématique

L'écriture des équations cinématiques est la première étape de la résolvabilité du robot qui peut résulter soit de la résolution du problème cinématique direct, soit de celle du problème cinématique inverse. Dans l'un ou l'autre cas, cette résolution doit permettre de concevoir l'algorithme de génération de la trajectoire à partir duquel sera basée la commande du bras.

2.3.1 Le problème cinématique direct

Le problème fondamental de la cinématique directe est la formulation des équations qui permettent d'exprimer les coordonnées spatiales de position et d'orientation de l'organe effecteur du robot en fonction des variables articulaires.

Si nous notons **X** le vecteur des coordonnées spatiales et **q** le vecteur des variables articulaires, l'équation cinématique directe s'écrit alors:

$$\mathbf{X} = \mathbf{F}(\mathbf{q}) \qquad (2.8)$$

F étant généralement une fonction non linéaire.

La résolution de cette équation consiste à calculer à chaque instant la position et l'orientation de l'organe terminal pour un ensemble de déplacements articulaires donné; en d'autres termes, le vecteur **q** étant connu, on détermine le vecteur **X** correspondant.

Pour les robots ayant une géométrie simple (2 à 3 axes), il est assez aisé d'utiliser l'approche géométrique pour exprimer les coordonnées spatiales en fonction des coordonnées articulaires.

Pour les structures plus complexes, l'expression explicite de l'équation cinématique directe à partir de la matrice de transformation **T** est toujours envisageable. Cependant, elle est assez longue à établir à cause des nombreuses fonctions trigonométriques qui sont impliquées. Néanmoins, la résolution du problème cinématique par la cinématique directe a l'avantage de toujours aboutir à une solution unique, d'un point de vue mathématique.

2.3.2 Le problème cinématique inverse

L'activation des robots implique généralement le déplacement de l'organe terminal vers une position donnée et avec une orientation précise. Ainsi, seules les coordonnées spatiales de position et d'orientation sont souvent connues; il faut alors calculer les déplacements devant être imposés aux articulations pour pouvoir atteindre la position désirée. Cela constitue le problème cinématique inverse dont la résolution consiste à trouver les solutions de l'équation.

$$\mathbf{q} = \mathbf{F}^{-1}(X) \tag{2.9}$$

Deux approches sont utilisées pour résoudre cette équation; une approche analytique et une approche numérique. L'une ou l'autre ne conduit à des solutions que si la structure du robot le permet. Cependant, la résolution par des méthodes numériques se prête, souvent, plus facilement au traitement en temps réel.

2.4 Difficultés de la cinématique inverse

Deux difficultés majeures apparaissent dans la résolution du problème cinématique inverse; la complexité des fonctions et la non unicité des solutions.

L'équation (2.9) qui décrit le problème inverse se traduit en réalité par un système de plusieurs équations généralement fortement non linéaires, impliquant des

fonctions trigonométriques de plusieurs variables. Cela donne lieu à des algorithmes de cinématique inverse très complexes, ce qui constitue un sérieux problème lorsqu'on désire réduire le temps de calcul et surtout opérer en temps réel.

D'autre part, à cause de la non linéarité des équations, l'équation cinématique inverse admet souvent un nombre de solution supérieur à 1; et même si toutes les solutions ne sont pas nécessairement accessibles, leur multiplicité (lorsqu'elles sont en nombre fini) impose un traitement supplémentaire en fonction des contraintes ou des incompatibilités avec les possibilités mécaniques du bras, ce qui a pour effet de rallonger le temps de calcul.

Quoique les méthodes classiques de résolvabilité des systèmes articulés utilisent toutes, d'une manière générale, des algorithmes de cinématique inverse avec succès, elles imposent néanmoins aux constructeurs de mettre en oeuvre des algorithmes d'interpolation très performants et d'utiliser des calculateurs puissants et coûteux afin de pouvoir opérer en temps réel, ou tout au moins de minimiser le temps de calcul.

Face à toute cette complexité induite par la cinématique inverse, une question s'impose à l'esprit : pourquoi n'essaie-t-on pas de générer les trajectoires sans passer par des algorithmes de cinématique inverse ? Pourquoi n'essaie-t-on pas de résoudre le problème cinématique des systèmes articulés en se basant sur la cinématique directe pour laquelle la solution est toujours unique, quelle que soit la structure considérée ?

CHAPITRE III

FORMULATION D'UNE MÉTHODE DE RÉSOLUTION ÉLIMINANT LA CINÉMATIQUE INVERSE

3.1 Principe de la méthode

Lorsqu'on commande un bras de robot, le problème se présente de la manière suivante : le bras occupe une configuration quelconque mais possible de son espace de travail et on voudrait positionner son organe effecteur en un point de cet espace avec une orientation précise par rapport au référentiel de base. Il s'agit donc de formuler, de façon très explicite, les relations qui lient les coordonnées spatiales et les coordonnées articulaires.

C'est cette formulation qui fait l'objet des algorithmes de cinématique inverse dont la complexité nous a amenés à rechercher une solution plus simple à la résolvabilité des robots articulés.

Le point de départ est le fait qu'une commande appliquée à une articulation du robot ne peut avoir que trois effets :

1) lui imprimer soit une rotation dans un sens défini positif, soit un allongement pour les articulations prismatiques;

2) lui imprimer soit une rotation dans le sens négatif, soit un raccourcissement pour les articulations prismatiques;

3) la maintenir immobile.

C'est ainsi que nous introduisons la notion de pas articulaire en émettant comme hypothèse que toute commande appliquée à une articulation produit au niveau de celle-ci un pas articulaire. En fixant à l'avance les différents pas articulaires possibles, nous définissons un nombre fini de commandes articulaires; donc, pour un nombre donné d'articulations, nous pouvons établir un nombre fini de combinaisons de ces commandes articulaires.

Ainsi, toutes les possibilités de commandes étant préalablement établies, nous éliminons d'un coup le problème le plus complexe de la cinématique inverse qu'est le calcul des commandes articulaires.

La génération de la trajectoire du robot devient une génération de petits déplacements, adéquatement sélectionnés et résultant d'un choix successif de combinaisons appropriées.

3.2 Définition des paramètres dynamiques

3.2.1. Définition de pas articulaire

Lorsqu'on applique une commande à une articulation de robot, elle effectue une rotation si c'est une articulation rotoïde, une translation si c'est une articulation prismatique, ou encore elle reste dans sa position actuelle.

La rotation peut s'effectuer dans un sens défini positif ou dans le sens inverse et la translation peut être un allongement ou un raccourcissement. Partant de cette constatation, nous pouvons donc définir un pas articulaire qui est une variation élémentaire de la variable articulaire. Ainsi, pour réaliser une variation articulaire de grande amplitude, il faut effectuer une succession de pas articulaires. Si nous désignons par "STEP" ce pas articulaire, il en résulte que :

- une variation dans le sens positif vaut " + STEP"
- une variation dans le sens négatif vaut " - STEP"
- une variation nulle vaut " 0 STEP"

3.2.2 Pas normalisé d'incrément

L'introduction d'un pas articulaire suggère évidemment l'utilisation d'une méthode de commande incrémentale. En effet, en fixant le pas articulaire, toute variation articulaire devient un incrément qui peut être négatif, positif ou nul.

Le pas de base utilisé pour l'analyse des petits déplacements de l'organe terminal est normalisé à un. Au cours de l'analyse du problème cinématique pour un bras donné, ce pas d'incrément normalisé, choisi de manière heuristique, peut s'avérer non optimal. On utilise alors un facteur k de dénormalisation tel que la valeur finale du pas est :

$$k.\ \text{STEP}$$

Le facteur k est déterminé par l'analyse du déplacement élémentaire et choisi selon les exigences sur la tolérance en positionnement et en orientation de l'outil.

3.2.2.1 Cartéristique d'incrément

Le pas d'incrément étant défini, chaque fois qu'une articulation reçoit une commande, elle doit donc se mouvoir dans un sens ou dans l'autre selon sa caractéristique d'incrément. La caractéristique d'incrément est par conséquent la valeur algébrique du STEP et prend les valeurs :

"+ STEP", "- STEP" ou "0 STEP".

3.2.3 Notion de variable trinaire

Par analogie à une variable binaire qui n'admet que deux valeurs, nous appelons variable "trinaire" une variable définie dans la base trois, à laquelle on ne peut attribuer que trois valeurs possibles. Ainsi, pour un nom-

bre donné n de variables trinaires, nous pouvons établir 3^n combinaisons différentes de ces trois valeurs.

Si nous considérons à présent les articulations d'un bras de robot auxquelles on associe un pas articulaire, chacune d'elle possède trois caractéristiques d'incrément et on peut donc lui associer comme une variable trinaire.

3.3 Formulation de la méthode

Soit le problème de commander un bras manipulateur dont la structure et le nombre d'articulations sont connus. À partir des dimensions du bras et selon la tolérance désirée, nous évaluons de manière heuristique, un pas normalisé d'incrément qui peut être réajusté par la suite. Par exemple, considérons un bras articulé, géométriquement tendu de façon à former une droite; on peut penser à priori que le déplacement élémentaire maximum de son extrémité a lieu si toutes les articulations effectuent simultanément une rotation dans le même sens. Ceci permet, en fixant cette "valeur maximale" du déplacement élémentaire *del*, de se donner un premier pas d'incrément que l'on normalise à un.

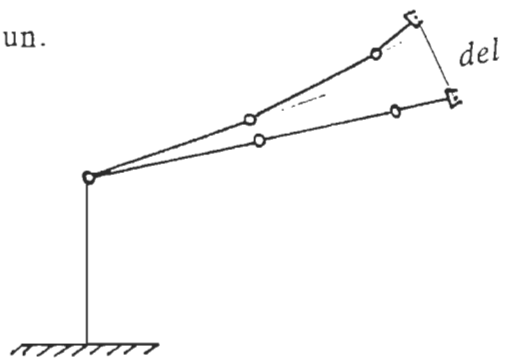


Figure 3.1 : Déplacement élémentaire de l'organe effecteur

Les variables articulaires étant des variables trinaires, toutes les articulations de même nature admettent les mêmes possibilités de caractéristiques d'incrément. L'activation du manipulateur revient donc à envoyer au niveau de chaque articulation la caractéristique appropriée, qui permet d'obtenir à l'extrémité du bras, le déplacement élémentaire désiré.

Une commande apparaît donc comme une combinaison de variables trinaires. La trajectoire est générée par une succession adéquate de déplacements élémentaires résultant d'une séquence appropriée de combinaisons trinaires.

Si nous commandons un bras possédant n articulations, nous avons un nombre fini 3^n de combinaisons de variables trinaires et par conséquent 3^n commandes possibles. Toutes les possibilités de commande étant fixées à l'avance, les valeurs des coordonnées articulaires sont toujours connues. Les coordonnées spatiales se calculent avant chaque incrément à partir de l'équation de cinématique directe.

$$\mathbf{X} = F(\mathbf{q})$$

Nous éliminons ainsi la nécessité et la complexité du calcul des commandes devant être appliquées aux articulations comme l'imposent les algorithmes de cinématique inverse.

Toute l'analyse du problème cinématique est basée sur le déplacement élémentaire de l'organe terminal du bras. Le principe du pas articulaire implique un fonc-

tionnement par petits déplacements; les équations de base de la cinématique directe ne peuvent donc pas être utilisées intégralement. Elles sont au préalable ramenées à des variations de coordonnées articulaires et spatiales afin de simplifier davantage les algorithmes de génération de la trajectoire.

3.4 Algorithmes de génération des trajectoires

3.4.1 Algorithme de validation de la méthode

Considérons à nouveau un bras ayant n articulations et notons $Q_i = [c_1\ c_2\ \ldots\ c_n]$, un vecteur de commande destiné à ce bras. Q_i est une commande parmi 3^n commandes tandis que c_j est une caractéristique d'incrément destinée à l'articulation numérotée j. Le vecteur des caractéristiques d'incrément est par conséquent un vecteur de variations articulaires.

Pour effectuer l'analyse du manipulateur, nous devons d'abord établir les relations mathématiques qui lient la position et l'orientation du bras aux variables articulaires, ce qui amène le concepteur à définir un référentiel de base fixe (souvent lié au châssis du robot) ainsi qu'un référentiel outil lié au centre de l'organe terminal. Dans notre étude, les coordonnées spatiales (position et orientation) seront toujours exprimées par rapport au référentiel de base noté R_o.

3.4.1.1 Mise en équation

Notons $X(R_o)$le vecteur caractérisant la position et l'orientation de l'organe terminal dans le référentiel de base R_o et Θ le vecteur des variables articulaires. Le point de départ de notre analyse est la relation cinématique directe :

$$X(R_o) = F(q) \tag{3.1}$$

telle que

$$X_1 = F_1 (q_1\ q_2 \ldots q_n)$$

$$X_2 = F_2 (q_1\ q_2 \ldots q_n)$$

$$X_m = F_m (q_1\ q_2 \ldots q_n)$$

Soit P, le centre de l'organe terminal dont la position et l'orientation dans le référentiel de base sont données par P_x, P_y, P_z et Φ_x, Φ_y, Φ_z respectivement.

Pour de petits accroissements, la fonction $F : R^{nx1} \text{-----} > R^{mx1}$ est considérée linéarisable. Par une approximation du premier ordre des variations, nous obtenons : $\Delta X(R_0) = [\frac{\partial F}{\partial q}] \Delta q$ (3.2)

où $J = [\frac{\partial F}{\partial q}]$ est la matrice Jacobienne.

Le vecteur des accroissements spatiaux est: $\Delta X(R_0) = [\Delta P_x\ \Delta P_y\ \Delta P_z\ \Delta \Phi_x\ \Delta \Phi_y\ \Delta \Phi_z\ ]^t$

Le vecteur des accroissements articulaires est : $\Delta q = [\Delta q_1 \Delta q_2 \ldots \Delta q_n]^t$

Mathématiquement, la nouvelle position et la nouvelle orientation du centre de l'organe terminal sont données par les relations

$$\begin{aligned} P_{x+} &= P_x + \Delta P_x \\ P_{y+} &= P_y + \Delta P_y \\ P_{z+} &= P_z + \Delta P_z \\ \Phi_{x+} &= \Phi_x + \Delta \Phi_x \\ \Phi_{y+} &= \Phi_y + \Delta \Phi_y \\ \Phi_{z+} &= \Phi_z + \Delta \Phi_z \end{aligned} \qquad (3.3)$$

Ainsi, lorsqu'on fait subir aux articulations une variation Δq, on obtient un déplacement élémentaire *del* de l'organe terminal tel que :

$$del = \| OP_+ - OP \|$$

$$del = \sqrt{(\Delta P_x)^2 + (\Delta P_y)^2 + (\Delta P_z)^2} \qquad (3.4)$$

C'est l'étude du déplacement élémentaire *del* qui permet de valider la méthode et le pas articulaire choisi pour le bras manipulateur considéré.

3.4.1.2 Principe de validation

Le déplacement élémentaire *del*, obtenu à partir du point *P* résulte d'une commande Q_i, définie plus haut, appliquée au bras. Nous avons par conséquent, pour un bras possédant *n* articulations, 3^n possibilités de déplacements élémentaires autour d'un point *P*. La trajectoire du centre de l'organe terminal est réalisée par des choix successifs adéquats de commande Q_i. Cependant, la précision de

cette trajectoire est étroitement liée à la finesse du déplacement élémentaire. De ce fait, la méthode ne peut être utilisable que si elle répond aux critères de précision et de tolérance fixés pour le robot considéré. Pour permettre à tout utilisateur de la méthode de s'assurer de son applicabilité pour un bras donné, nous avons développé un "logiciel" de vérification et de validation.

C'est un logiciel interactif dont les objectifs sont de permettre à l'utilisateur de vérifier si le déplacement élémentaire maximum satisfait aux exigences de tolérance fixées et de déterminer, si besoin est, le facteur de dénormalisation adéquat.

Son principe est de rendre discret l'espace accessible du robot, de calculer toutes les possibilités de déplacements élémentaires, d'identifier le plus grand déplacement élémentaire et d'ajuster le coefficient de dénormalisation en conséquence.

3.4.1.3. Description de l'algorithme

Le logiciel de vérification et de validation est conçu pour un bras de robot ayant six articulations ou moins; cependant il peut être étendu à un plus grand nombre d'axes. L'algorithme, quoique simple, nécessite une préparation laborieuse.

Au départ, nous établissons la matrice globale de transformation pour un bras à six axes :

$$T_o^6 = A_o^1 A_1^2 A_2^3 A_3^4 A_4^5 A_5^6$$

Dans cette expression, chaque paramètre garde sa forme littérale pour ne pas nuire à la généralité.

De cette matrice nous extrayons, par différentiation des expressions littérales, la matrice Jacobienne de position $[J_p]$ qui lie les variations des coordonnées spatiales de position aux variations des coordonnées articulaires.

$$\begin{pmatrix} \Delta P_x \\ \Delta P_y \\ \Delta P_z \end{pmatrix} = [J_p] \begin{pmatrix} \Delta q_1 \\ \Delta q_2 \\ \Delta q_3 \\ \Delta q_4 \\ \Delta q_5 \\ \Delta q_6 \end{pmatrix}$$

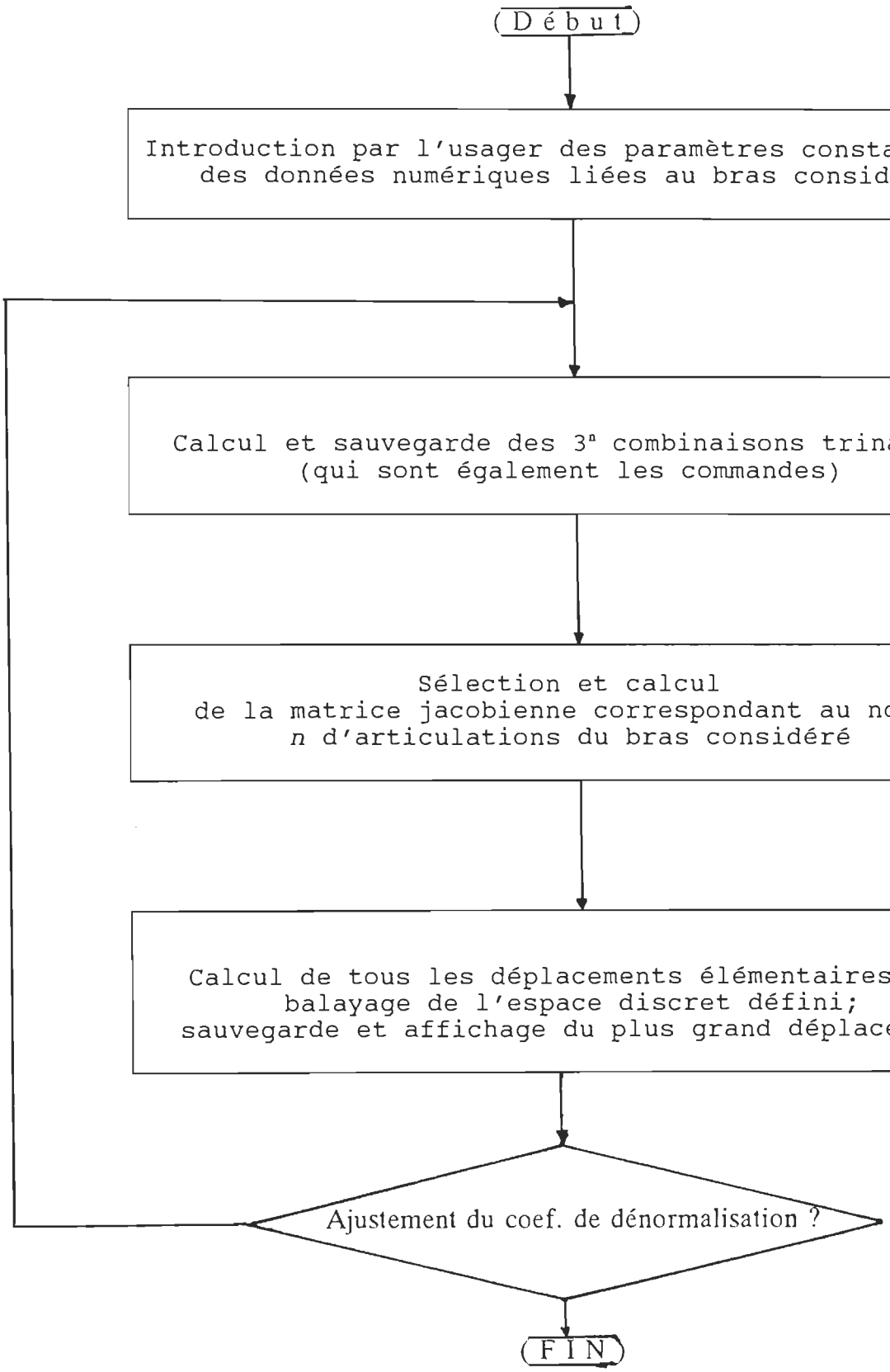
Le vecteur Δq des variations articulaires étant équivalent au vecteur de commande Q_i, à chaque commande correspond un vecteur ΔP_i des variations des coordonnées spatiales de position tel que :

$$\Delta P_i = \begin{pmatrix} \Delta P_{xi} \\ \Delta P_{yi} \\ \Delta P_{zi} \end{pmatrix}$$

Nous pouvons alors calculer pour chaque point de l'espace accessible discret, les 3^n possibilités de déplacement élémentaire, soit :

$$del_i = \sqrt{(\Delta P_{x_i})^2 + (\Delta P_{y_i})^2 + (\Delta P_{z_i})^2} \qquad i = 1, \ldots n$$

La quantification de l'espace revient à l'utilisateur qui, selon la structure du bras qu'il étudie, doit se donner un pas de balayage pour les articulations rotoïdes et/ou un pas de balayage pour les articulations prismatiques. Les sous-programmes dépendent du nombre d'articulations. Une fois que le nombre d'articulations et les paramètres du bras ont été introduits, le module correspondant au nombre n d'articulations est sélectionné pour calculer la matrice jacobienne adéquate ainsi que tous les déplacements élémentaires de l'espace discret accessible. A la fin du balayage, seul le plus grand déplacement est sauvegardé. S'il s'avère satisfaisant, le pas articulaire est conservé; sinon, ce dernier doit être multiplié par un facteur de dénormalisation. L'opération se poursuit jusqu'à l'obtention d'un pas articulaire qui satisfait aux exigences fixées. L'algorithme du logiciel peut donc être schématisé par l'organigramme suivant :


Début
Introduction par l'usager des paramètres constants et des données numériques liées au bras considéré.
Calcul et sauvegarde des 3ª combinaisons trinaires (qui sont également les commandes)
Sélection et calcul de la matrice jacobienne correspondant au nombre n d'articulations du bras considéré
Calcul de tous les déplacements élémentaires par balayage de l'espace discret défini; sauvegarde et affichage du plus grand déplacement
Ajustement du coef. de dénormalisation ?
FIN

La méthode est valide pour une structure donnée dès que le plus grand déplacement élémentaire et le pas d'incrément définitif sont compatibles avec la tolérance fixée et les possibilités mécaniques des servomécanismes respectivement.

3.4.2 Génération de la trajectoire

D'une manière générale, l'ensemble des articulations d'un bras manipulateur usuel peut être séparé en deux groupes : un groupe d'articulations lié au positionnement et un groupe lié à l'orientation. Pour une large classe de manipulateurs à n degrés de liberté, les trois premières articulations constituent généralement le bras et donc, servent au positionnement, tandis que les n-3 articulations restantes constituent le poignet et servent à l'orientation.

Dans l'étude de la génération des trajectoires, il est donc naturel de décomposer le mouvement global du bras en positionnement et orientation. Le vecteur $\mathbf{P}_o$ qui décrit l'extrémité de l'organe terminal par rapport au référentiel de base peut ainsi être décomposé en vecteur position $\mathbf{P}_o^a$ et en vecteur orientation $\mathbf{P}_o^w$ tel que :

$$\mathbf{P}_o = \mathbf{P}_o^a + \mathbf{P}_o^w \tag{3.6}$$

(Voir figure 3.2)

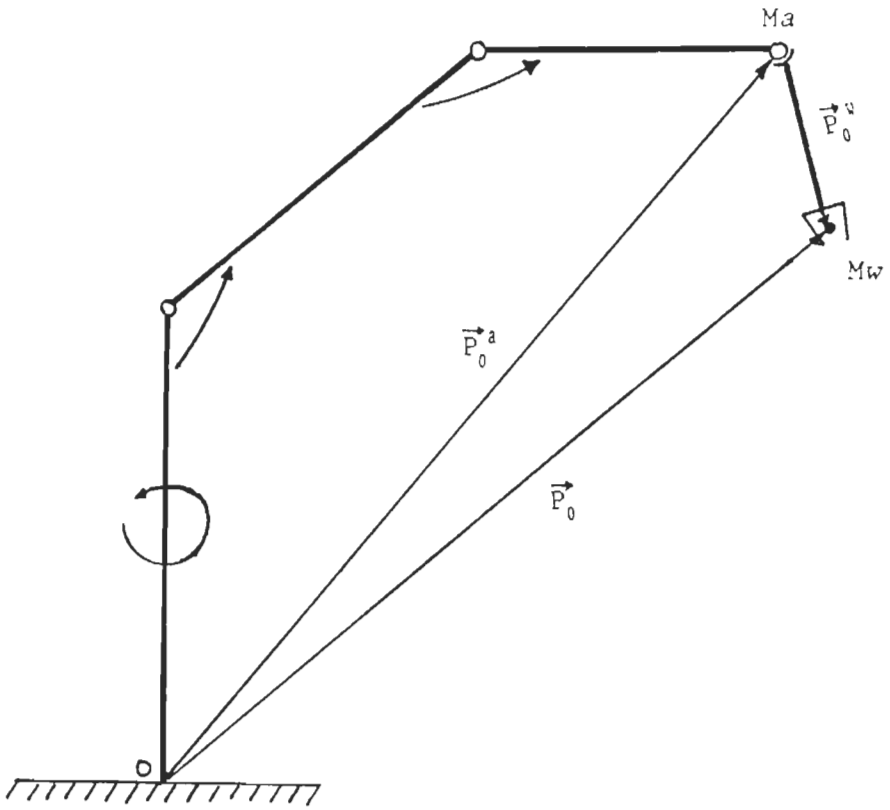


Figure 3.2 : Vecteur position et vecteur orientation

Le point M*a* représente l'extrémité du bras, tandis que le point M*w* représente l'extrémité du poignet correspondant au centre de l'outil. De nombreux bras actuels sont munis d'un poignet sphérique (poignet triaxial à axes concourants) qui offre une plus grande flexibilité dans l'orientation de l'outil. De ce fait, nous adoptons pour notre étude et pour les simulations, une structure articulée à trois axes de rotation, munie d'un poignet sphérique.

Cependant, le principe reste le même pour toutes les structures de poignets à trois axes, même si quelques ajustements mathématiques peuvent s'avérer nécessaires, notamment dans l'établissement de la matrice de transformation.

L'étude peut être faite pour une structure quelconque selon le même principe; cependant, pour être plus concis, nous appliquons la méthode à une structure particulière qui est la structure articulée à six axes.

Si nous notons (P_{ax}, P_{ay}, P_{az}) les coordonnées du vecteur $\mathbf{P}_o^a$ et (P_{wn}, P_{wy}, P_{wz}) les coordonnées du vecteur $\mathbf{P}_o^w$, il est possible de les décrire dans le référentiel de base (fig. 3.3) en partant de la matrice de transformation globale T.

$$T = \begin{bmatrix} n_x & o_x & a_x & P_x \\ n_y & o_y & a_y & P_y \\ n_z & o_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

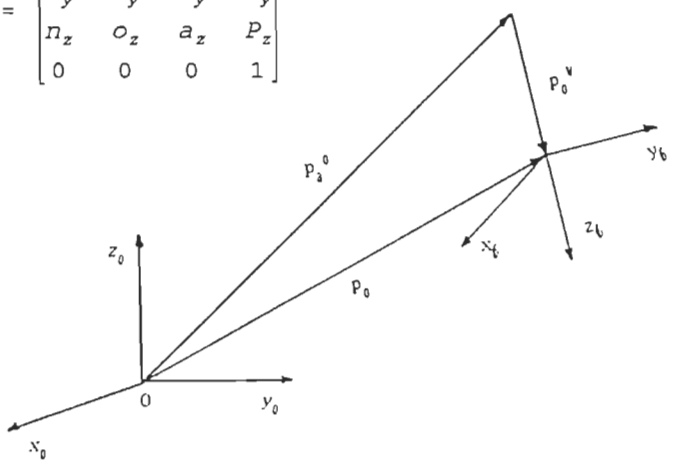


Figure 3.3 : Description dans R_o des vecteurs position et orientation

Pour le manipulateur à six axes que nous avons considéré, la matrice T peut être réécrite sous la forme :

$$T = [T_o^a]\,[T_a^w] \tag{3.7}$$

avec $T_o^a = A_o^1\, A_1^2\, A_2^3$ matrice de transformation du bras

et $T_a^w = A_3^4\, A_4^5\, A_5^6$ matrice de transformation du poignet

on obtient ainsi :

$$T_o^a = \begin{bmatrix} R_o^a & p_o^a \\ 0 & 1 \end{bmatrix} \qquad T_a^w = \begin{bmatrix} R_a^w & p_a^w \\ 0 & 1 \end{bmatrix}$$

En réécrivant (3.7) on a :

$$\begin{bmatrix} R & p_o \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_o^a & p_o^a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_a^w & p_a^w \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_0^a\, R_a^w & p_o^a + R_0^a\, p_a^w \\ 0 & 1 \end{bmatrix}$$

d'où $\mathbf{p}_o = \mathbf{p}_o^a + R_o^a \mathbf{p}_a^w$. Le vecteur $\mathbf{p}_a^w$ étant directement obtenu de la matrice T_a^w, on peut donc calculer $\mathbf{p}_o^w = R_o^a \mathbf{p}_a^w$.

La position de l'extrémité du bras est donc donnée par

$\mathbf{p}_o^a = \mathbf{p}_o - \mathbf{p}_o^w$

soit $\quad p_{ax} = p_x - p_{wn}$

$\quad p_{ay} = p_{yx} - p_{wy}$

$\quad p_{az} = p_z - p_{wz}$

pour un poignet sphérique, le vecteur $\mathbf{p}_o^w$ veut :

$$p_o^w = \begin{bmatrix} p_{wx} \\ p_{wy} \\ p_{wz} \end{bmatrix} = \begin{bmatrix} d_6\, a_x \\ d_6\, a_y \\ d_6\, a_z \end{bmatrix} \tag{3.8}$$

ce qui donne :
$$\begin{aligned} p_{ax} &= p_x - d_6\, a_x \\ p_{ay} &= p_y - d_6\, a_y \\ p_{az} &= p_z - d_6\, a_z \end{aligned}$$

La matrice de transformation T ayant été calculée, les cosinus directeurs a_x, a_y, a_z sont connus. Ainsi la donnée des vecteurs $\mathbf{p}_o$ et $\mathbf{p}_o^a$ définit entièrement le point de la trajectoire et l'orientation du poignet.

Partant de cette analyse, nous avons développé une méthode itérative pour la génération de la trajectoire où le mouvement du bras est décomposé pour chaque point en positionnement puis orientation.

3.4.2.1 Positionnement du bras

Le positionnement du bras n'implique que les trois premières articulations du robot. Nous établissons donc la matrice Jacobienne de position à partir du vecteur $\mathbf{p}_o^a$ par première approximation des variations des composantes, ce qui nous donne :

$$\begin{bmatrix} \Delta p_{axi} \\ \Delta p_{ayi} \\ \Delta p_{azi} \end{bmatrix} = [J_{poi}] \begin{bmatrix} \Delta q_1 \\ \Delta q_2 \\ \Delta q_3 \\ \Delta q_4 \\ \Delta q_5 \\ \Delta q_6 \end{bmatrix} \qquad (3.9)$$

Le passage d'une position courante à une position visée se fait après plusieurs itérations. Après chaque itération, les variations ΔP_{axi}, ΔP_{ayi} et ΔP_{azi} s'ajoutent aux valeurs des coordonnées du point courant, jusqu'à ce que la position visée soit atteinte.

Le vecteur Δq est un vecteur de combinaisons trinaires qui à chaque itération est sélectionné de manière à rapprocher le bras de la position visée tout en minimisant les variations de l'orientation.

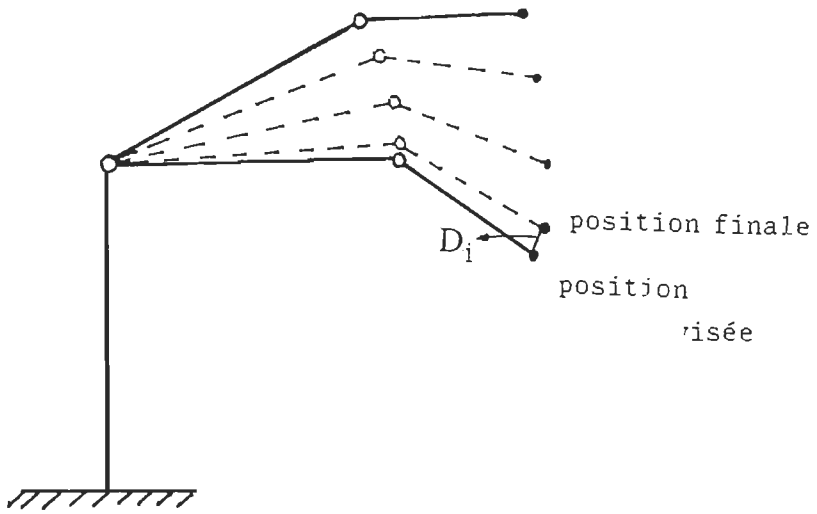


Figure 3.4 : Positionnement du bras

Cependant, les variations que subissent les articulations d'orientation au cours du positionnement n'influencent pas ce dernier. Aussi, le critère de sélection des combinaisons trinaires n'est pas très rigoureux quant à l'immobilité des articulations d'orientations. Le positionnement est terminé lorsque la distance D_i (fig. 3.4) entre la

position finale et la position visée est inférieure à une valeur ϵ_p fixée selon la précision et la tolérance désirées.

3.4.2.2 Orientation du poignet

Lors de l'orientation du poignet, seules les n-3 dernières articulations doivent subir des variations. Supposons à présent l'extrémité du bras positionné et considérons l'orientation du poignet. Pour passer de la position courante à la position visée, le vecteur d'orientation doit se déplacer d'un angle β (fig. 3.5). D'après (3.8), d_6 étant la longueur totale du poignet, le vecteur $\mathbf{M_aM_w}$ est défini par la donnée des cosinus directeurs a_{xw}, a_{yw} et a_{zw} correspondant.

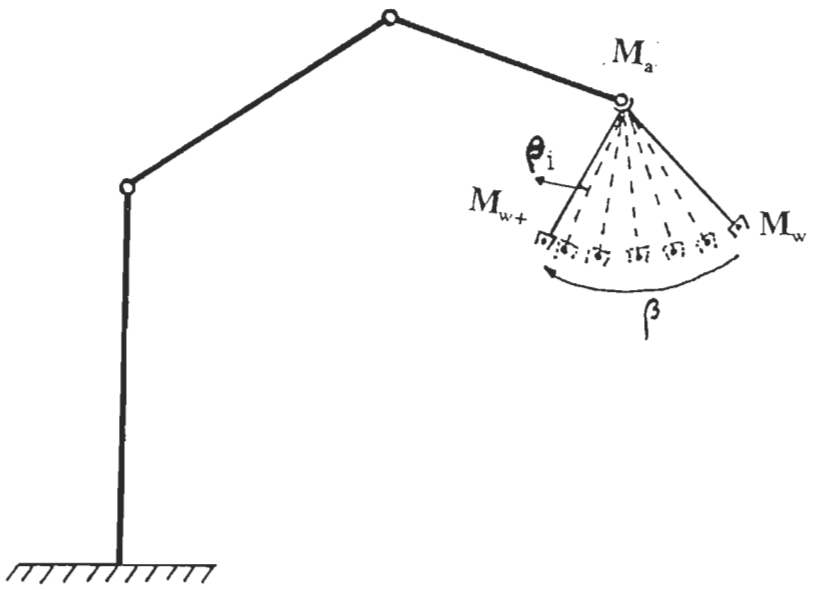


Figure 3.5 : Orientation du poignet

Nous avons donc :

$$M_a M_w = \begin{bmatrix} d_6 & a_{xw} \\ d_6 & a_{yw} \\ d_6 & a_{zw} \end{bmatrix} \quad et \quad M_a M_{w+} = \begin{bmatrix} d_6 & a_{xw+} \\ d_6 & a_{yw+} \\ d_6 & a_{zw+} \end{bmatrix}$$

qui sont connus. La rotation du vecteur $\mathbf{M_aM_w}$ produit un déplacement D de l'extrémité du poignet tel que :

$$D = \sqrt{d_6^2 [(a_{xw+} - a_{xw})^2 + (a_{yw+} - a_{yw})^2 + (a_{zw+} - a_{zw})^2]}$$

Connaissant D, nous pouvons évaluer l'expression de l'angle β. Ainsi, on a :

$$\sin\beta = \frac{D/2}{d_6} = \frac{1}{2} \frac{d_6}{d_6} \sqrt{[(a_{xw+} - a_{xw})^2 + (a_{yw+} - a_{yw})^2 + (a_{zw+} - a_{zw})^2]}$$

d'où (3.10)

$$\beta = \sin^{-1}\left[\frac{1}{2} [(a_{xw+} - a_{xw})^2 + (a_{yw+} - a_{yw})^2 + (a_{zw+} - a_{zw})^2]^{1/2}\right]$$

La variable à contrôler lors de l'orientation est donc l'angle β qu'il faut rendre nul; ce qui revient à rabattre progressivement le vecteur $\mathbf{M_aM_w}$ sur le vecteur $\mathbf{M_aM_{w+}}$.

Lors de l'orientation, le critère de sélection des combinaisons trinaires impose que les trois premières articulations ne subissent aucune variation. À chaque itération, la combinaison trinaire sélectionnée permet d'obtenir de nouveaux cosinus directeurs a_{xwi}, a_{ywi} et a_{zwi} tel que l'on ait :

$$\beta_i = \sin^{-1}\left[\frac{1}{2}\left[(a_{xwi} - a_{xw})^2 + (a_{ywi} - a_{yw})^2 + (a_{zwi} - a_{zw})^2\right]^{1/2}\right]$$

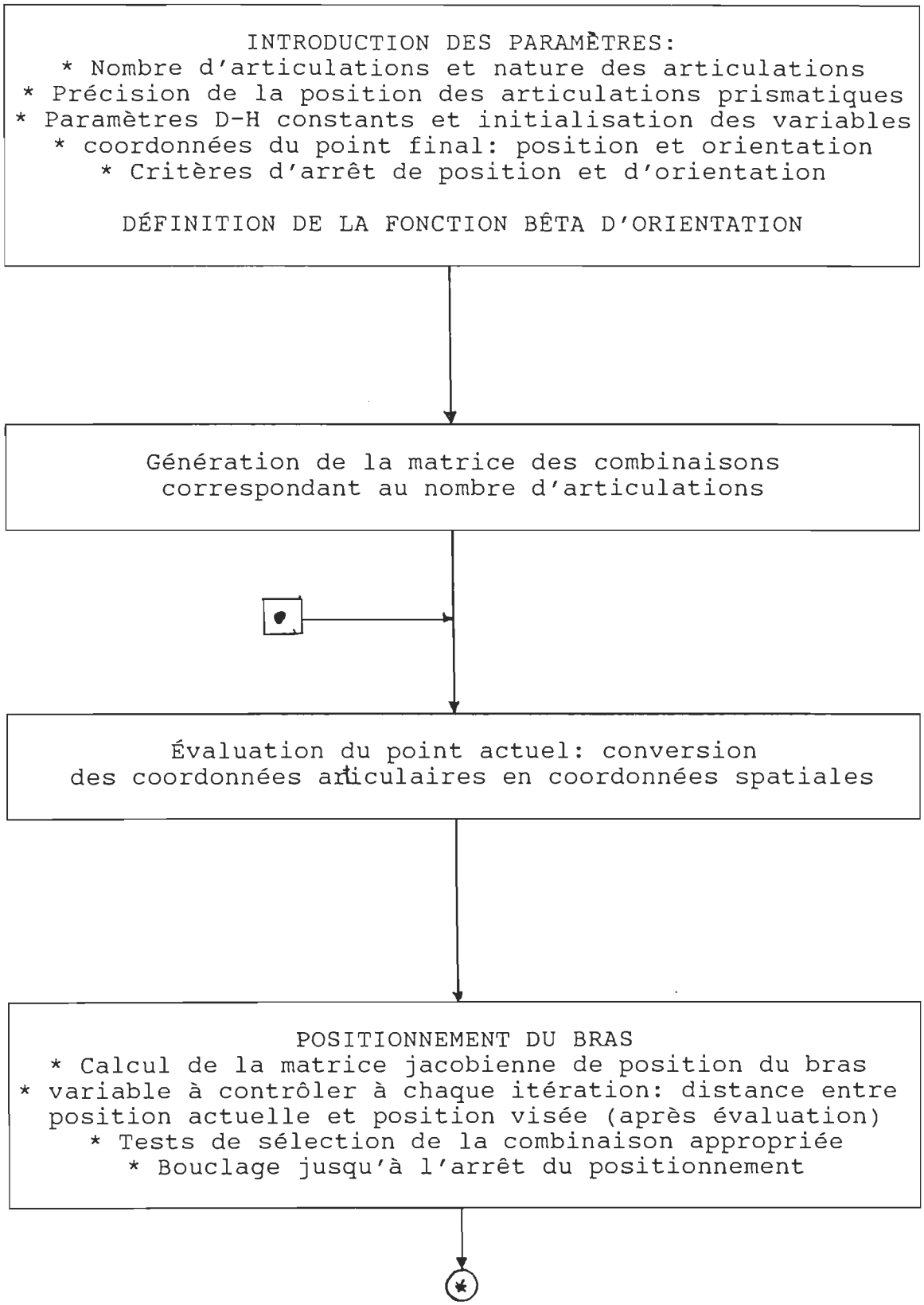
avec $\beta_i < \beta$; l'orientation est terminée lorsqu'on obtient $\beta_i < \epsilon_o$, ϵ_o étant fixée selon les critères de tolérance.

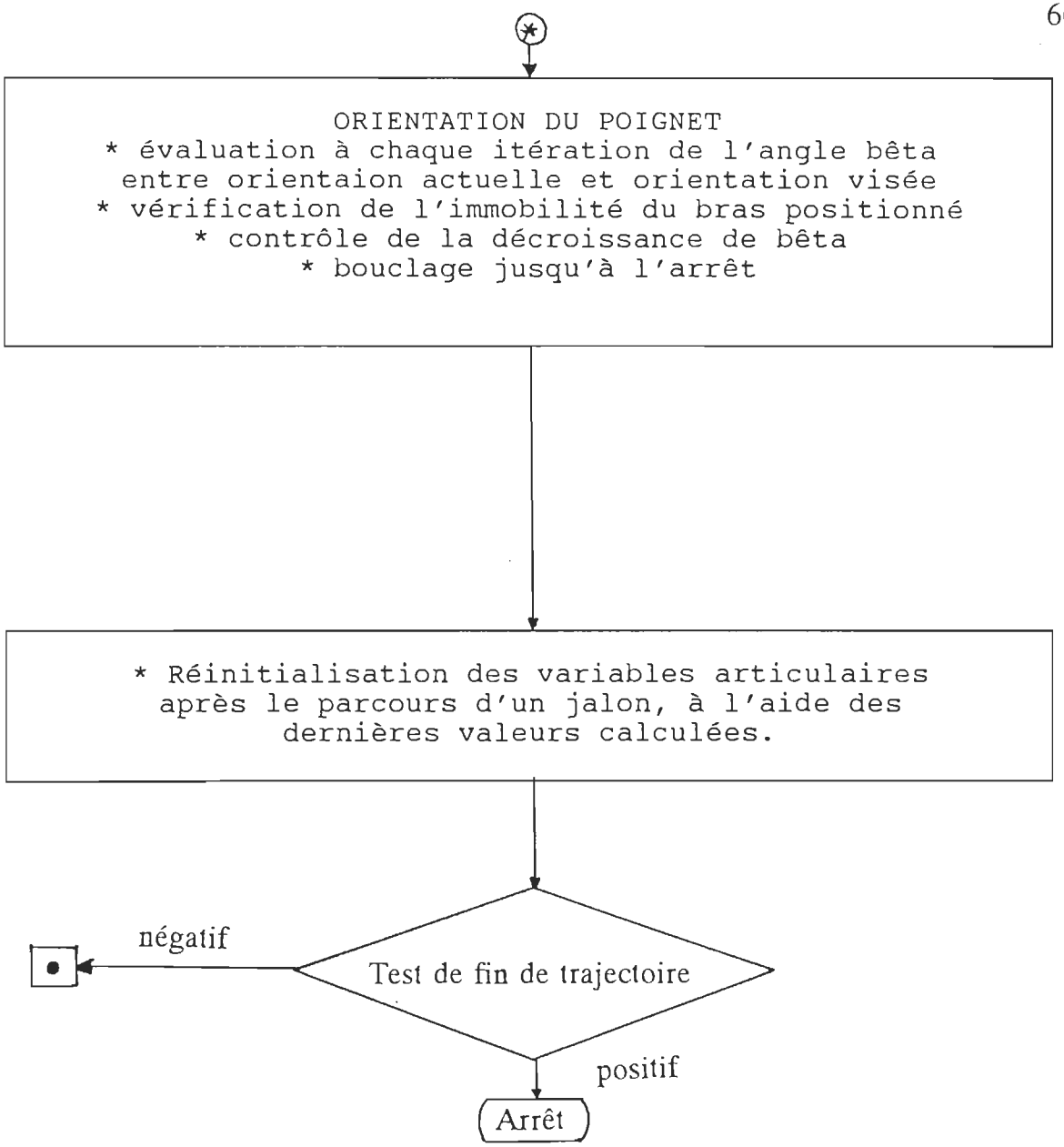
Ainsi, l'algorithme d'interpolation doit découper la trajectoire en jalons, et spécifier pour chaque point ses coordonnées dans le référentiel fixe, ainsi que les cosinus directeurs a_x, a_y et a_z correspondant à l'orientation du poignet désirée en ce point. Pour faire passer l'organe effecteur d'un jalon au jalon suivant, l'ordinateur ne **calcule** pas les commandes devant être appliquées aux articulations. Il doit sélectionner parmi les commandes déjà définies, celles qui sont le plus appropriées pour générer le mouvement désiré. Le parcours d'un jalon est réalisé après un certain nombre d'itérations. La précision de la courbe dépend du nombre de points spécifiés par l'interpolateur car chaque point est atteint avec une très bonne précision.

3.4.3 Description de l'algorithme de génération de la trajectoire

La méthode décrite se traduit par un algorithme simple dont les principales étapes sont:

- l'introduction des paramètres
- la génération de la matrice des combinaisons
- l'évaluation de la position et le calcul de la position visée.


INTRODUCTION DES PARAMÈTRES:
* Nombre d'articulations et nature des articulations
* Précision de la position des articulations prismatiques
* Paramètres D-H constants et initialisation des variables
* coordonnées du point final: position et orientation
* Critères d'arrêt de position et d'orientation
DÉFINITION DE LA FONCTION BÊTA D'ORIENTATION
Génération de la matrice des combinaisons correspondant au nombre d'articulations
Évaluation du point actuel: conversion des coordonnées articulaires en coordonnées spatiales
POSITIONNEMENT DU BRAS
* Calcul de la matrice jacobienne de position du bras
* variable à contrôler à chaque itération: distance entre position actuelle et position visée (après évaluation)
* Tests de sélection de la combinaison appropriée
* Bouclage jusqu'à l'arrêt du positionnement



Les tests de sélection de la combinaison appropriée peuvent être différents suivant le concepteur et suivant la structure du bras. Ils doivent cependant assurer la convergence de l'algorithme.

3.5 Stratégie de commande

3.5.1 Problèmes liés au contrôle de la vitesse

La méthode développée permet d'assurer un passage assez précis par tous les points spécifiés le long de la trajectoire. Cependant, au niveau de chaque articulation la caractéristique d'incrément est toujours exécutée de la même façon, à une vitesse déterminée par l'asservissement. Il n'est donc pas possible de contrôler la vitesse de déplacement d'une articulation, car elle dépend essentiellement de la fréquence des caractéristiques d'incrément non nulles et de même signe qui lui parviennent.

De ce fait, le seul contrôle de vitesse qui soit possible est celui de la vitesse moyenne de déplacement de l'organe terminal. En effet, lorsqu'une commande est sélectionnée, chaque articulation réagit et exécute la caractéristique d'incrément qui lui est désignée, puis verrouille son moteur. Le mouvement simultané des articulations produit alors un déplacement du centre de l'effecteur à une vitesse instantanée V_x qui ne dépend que de l'ensemble des asservissements. À cette vitesse V_x, nous associons un temps T_x qui inclut l'exécution du déplacement et le retour de l'information au microprocesseur (fig. 3.6).

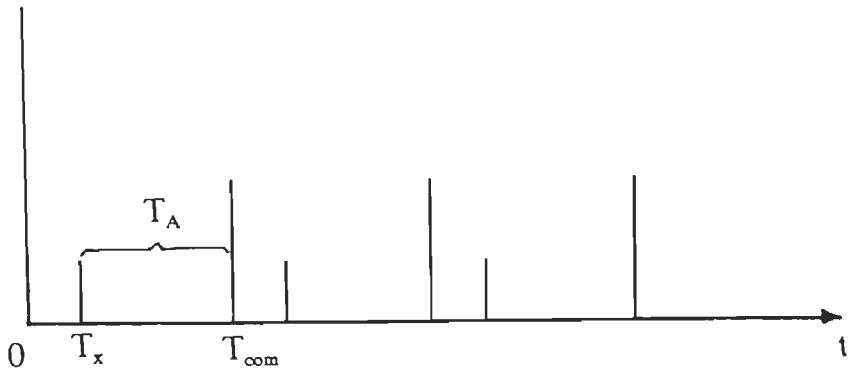


Figure 3.6 : Échantillonnage de la commande

Lors du parcours d'une trajectoire, la vitesse moyenne de déplacement de l'organe terminal sera maximale si les commandes choisies sont immédiatement exécutées les unes à la suite des autres. Ainsi, pour pouvoir faire varier cette vitesse moyenne de déplacement, il faudrait introduire un délai variable entre l'exécution de deux commandes successives. Son contrôle revient alors à mettre au point une stratégie permettant de réaliser ce délai variable.

3.5.2 Principe de contrôle de la vitesse

Pendant la poursuite de la trajectoire, les combinaisons trinaires sélectionnées comportent toujours au moins une commande non nulle. De ce fait, pour chaque combinaison **sélectionnée**, il y a toujours au moins une articulation qui entre en mouvement et impose par conséquent un déplacement au centre de l'effecteur . Si les combinaisons sont envoyées à une fréquence élevée, l'organe terminal va se mouvoir à une vitesse importante; pour réduire

cette vitesse, nous devons alors nous servir de la combinaison "nulle". Il s'agit de la combinaison pour laquelle tous les moteurs d'articulations restent verrouillés.

En effet, en exécutant un certain nombre de fois la combinaison nulle entre deux combinaisons non nulles, nous réalisons une période d'attente T_A qui freine le mouvement de l'organe terminal. La période d'échantillonnage des commandes T_{com} est alors égale à $(T_x + T_A)$ et varie selon la durée de T_A (voir figure 3.6). La vitesse moyenne de déplacement devient maximale lorsqu'on a $T_{com} = T_x$; le contrôle consiste alors à allonger ou réduire adéquatement la période T_A.

Le principe d'activation du bras est assimilable à celui du fonctionnement d'un moteur pas-à-pas. Dans la commande du moteur pas-à-pas, le rotor effectue un pas angulaire sous l'action d'une impulsion de courant envoyée sur un des bobinages du stator. L'alimentation des bobines selon une séquence appropriée permet de générer la rotation du moteur; chaque pas est exécuté en un temps minimum t_m qui est fonction des caractéristiques dynamiques du moteur. Une fois le pas effectué, le moteur attend l'impulsion de commande suivante; ainsi, la vitesse de rotation du moteur ne dépend que de la fréquence des impulsions. Cependant, malgré qu'on puisse la faire varier aisément, la période d'échantillonnage des commandes ne peut être inférieure au temps t_m.

Par ailleurs, même si la période d'échantillonnage est très grande, t_m ne varie pas.

Dans la commande du bras par la méthode développée, nous nous trouvons dans une situation comparable à celle du moteur pas-à-pas. Les caractéristiques dynamiques du système définissent le temps T_x qui est analogue à t_m, et par conséquent fixe. Cependant, contrairement au moteur pas-à-pas, il est moins aisé de faire varier la période, T_{com}, d'échantillonnage des commandes. Ceci explique la nécessité de concevoir un système de gestion de la période d'attente T_A à l'aide d'un ou plusieurs compteurs internes au processeur principal.

3.5.3 Stratégie de contrôle

Pour développer la stratégie, quelques hypothèses sont nécessaires. Ainsi, nous admettons que le niveau de technologie actuel permet la construction de moteurs à courant continu à réponse rapide et dont l'amortissement permet de réaliser les pas angulaires tel que définis. De plus, nous considérons que le courant dans chaque moteur est constant afin d'assurer à sa charge un couple constant. En d'autres termes nous prenons pour acquis que chaque asservissement est capable de répondre correctement aux commandes que nous lui appliquons.

En général, les robots actuels sont dotés d'un système de commande automatique qui asservit chaque articulation en position et en vitesse au moyen d'un contrôleur. Dans notre stratégie, le contrôleur lié à chaque articulation doit lui permettre d'effectuer ses caractéristiques d'incrément avec la meilleure précision, et en assurer la stabilité et l'amortissement. L'asservissement est essentiellement un asservissement de position.

Pour assurer le contrôle de la vitesse moyenne de déplacement de l'organe terminal, nous proposons d'adjoidre au système de commande une boucle de contrôle global pour laquelle on spécifie la vitesse moyenne désirée pour la trajectoire à parcourir.

En utilisant par ailleurs un registre du processeur principal comme compteur, la valeur qu'on y met détermine le nombre de fois qu'il est nécessaire d'envoyer aux articulations la combinaison nulle entre deux combinaisons non nulles et par conséquent, fixe la durée de T_A.

L'objectif de ce contrôle est de respecter, lors du parcours d'une trajectoire, la courbe trapézoïdale de vitesse qui présente une phase d'accélération, une phase de vitesse constante et une phase de décélération, tout en réalisant le long du parcours la vitesse moyenne de consigne. Il s'agit alors de modifier adéquatement la durée de T_A durant chacune des phases afin que le

déplacement du centre de l'effecteur évolue d'abord très vite, puis de manière à peu près constante et enfin plus lentement jusqu'à l'arrêt. La stratégie développée comporte deux volets dont un pour la commande manuelle et un pour la commande automatique.

3.5.3.1 Contrôle en mode manuel

Si on est amené à commander un bras en mode manuel, le contrôle de la vitesse est plus aisé. On établit au préalable une gamme de vitesses telle que chaque vitesse sélectionnée corresponde à une valeur qui est chargée dans le registre-compteur. L'opérateur peut alors choisir de commander les articulations individuellement ou encore de contrôler le déplacement de l'organe terminal. Quel que soit le mode de fonctionnement choisi, la trajectoire est parcourue à une vitesse constante jusqu'à ce que l'opérateur change lui-même la vitesse; il peut ainsi contrôler les accélérations et les décélérations.

3.5.3.2 Contrôle en mode automatique

Le contrôle en mode automatique implique deux grands problèmes que l'on rencontre toujours lorsqu'il s'agit de commander un système mécanique par un système électronique : il s'agit de l'informatique et de l'automatisme.

L'informatique implique tant la question matérielle que la question logicielle, car il s'agit d'un système de contrôle en temps réel. L'automatisme implique le système numérique et l'algorithme permettant d'assurer la sélection, la conversion et/ou l'envoi des signaux adéquats. L'algorithme de contrôle doit par conséquent gérer rigoureusement les différents signaux impliqués dans le parcours d'une trajectoire.

Si nous découpons la trajectoire en jalons, la méthode développée impose que chaque noeud soit atteint après un certain nombre d'itérations. Chaque noeud atteint correspond à un pourcentage de la trajectoire. L'interpolateur doit donc fournir à l'ordinateur maître le nombre de noeuds spécifiés de la trajectoire, la vitesse moyenne de parcours étant également précisée.

Ces données permettent à l'algorithme de contrôle de gérer les différentes phases de la courbe de vitesse (fig. 3.7).

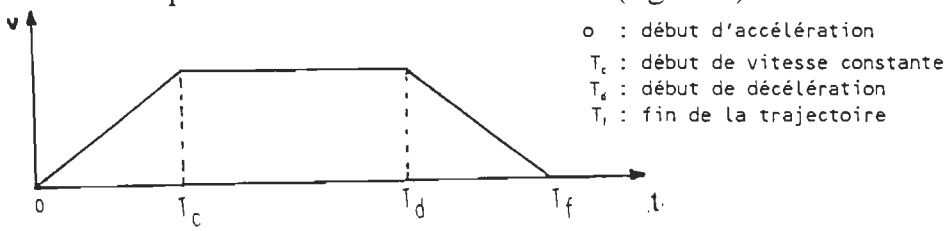


Figure 3.7 : Courbe de vitesse de l'organe terminal

Le choix de cette courbe de vitesse a pour but de permettre à l'organe terminal d'atteindre le point final sans heurts et sans oscillations.

Les noeuds qui sont les points spécifiés de la trajectoire ne servent, en réalité, que de repères pour permettre à l'algorithme de déterminer le début et la fin de chaque phase de vitesse.

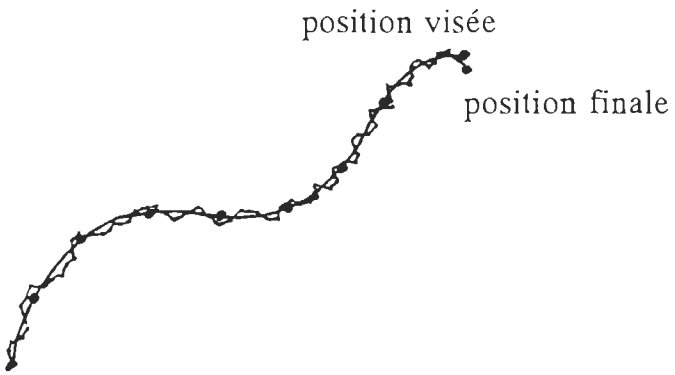


Figure 3.8 : mode de parcours d'une trajectoire

Ainsi, l'algorithme évalue à l'avance, en tenant compte du nombre de points de la trajectoire et de la vitesse moyenne exigée, la portion de la courbe sur laquelle se fait l'accélération, celle correspondant à la vitesse constante et celle correspondant à la

décélération. Pratiquement, les phases d'accélération et de décélération doivent être sensiblement égales.

La gestion des phases de vitesse se fait alors à l'aide du registre-compteur dans lequel sont chargées, selon un programme bien défini, les valeurs permettant d'atteindre les accélérations ou les décélérations désirées. La valeur chargée dans le registre-compteur détermine le nombre de commandes nulles qui doivent être envoyées aux actionneurs entre deux commandes non nulles.

Pour réaliser un système de commande performant, il est préférable que ce dernier soit constitué de plusieurs processeurs et même d'un coprocesseur. On peut donc schématiser le diagramme du système de commande comme suit :

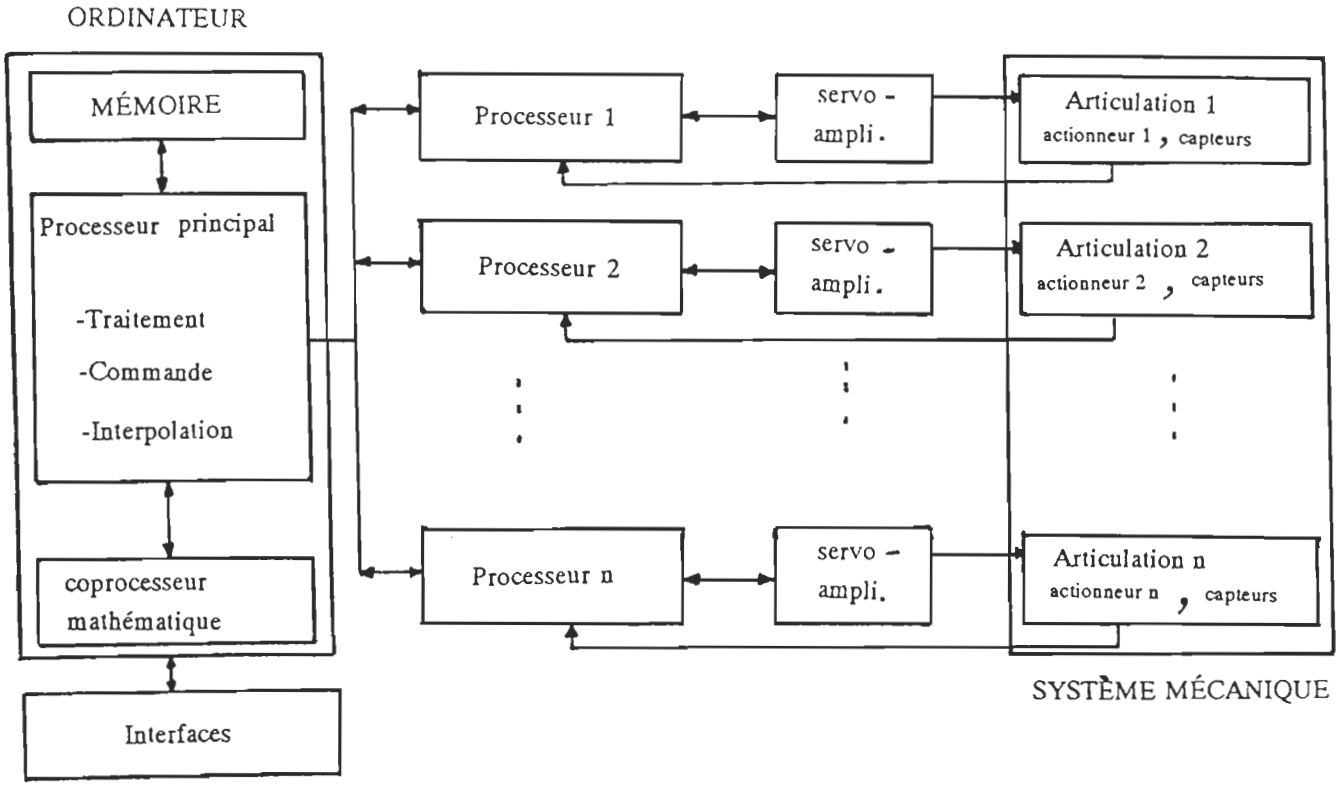


Figure 3.9 : diagramme d'un système de commande

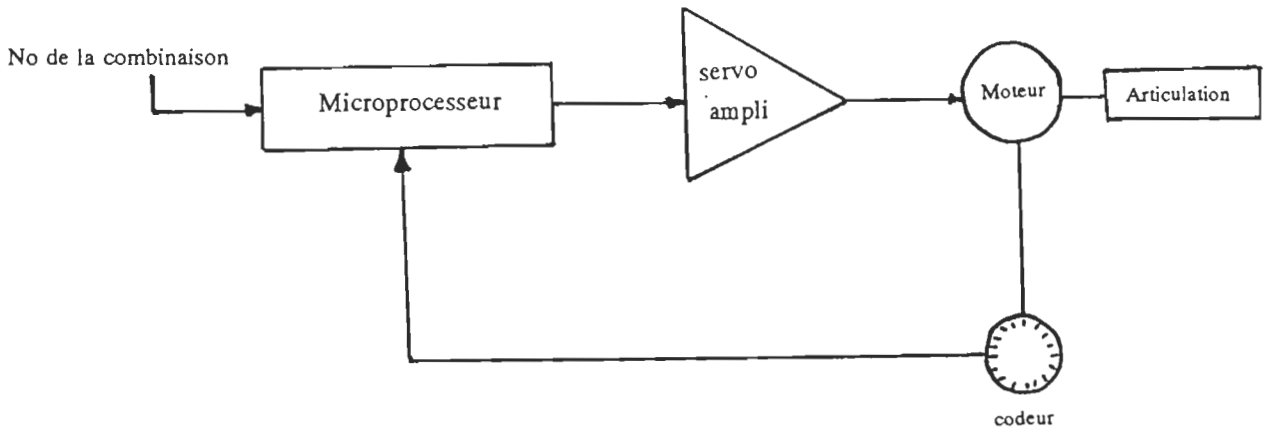


Figure 3.10 : Commande d'une articulation

Il faut noter cependant que même si l'ordre de grandeur reste le même, le nombre d'itérations pour parcourir deux jalons de même longueur est très souvent différent, surtout lorsque l'orientation est considérablement modifiée. Dès lors, pour réaliser un algorithme de contrôle efficace, il s'avère indispensable d'évaluer, en fonction des processeurs utilisés, le calibrage en vitesse du système; il s'agit d'évaluer le temps moyen nécessaire pour parcourir une certaine longueur de trajectoire, ainsi que le temps moyen nécessaire pour effectuer une certaine rotation du poignet. C'est en tenant compte de ces valeurs que l'on peut concevoir un algorithme capable de découper la trajectoire en phases de vitesses.

CHAPITRE IV

RÉSULTATS DE SIMULATIONS

4.1 Description des éléments de la simulation

Pour vérifier la théorie développée, nous avons effectué des simulations sur un bras de type PUMA dont les paramètres D-H sont donnés dans la table 4.1 :

Paramètres N° d'articulation	θ	$\alpha(^0)$	a(mm)	d(mm)
1	θ_1	- 90	0	660.4
2	θ_2	0	431.8	149.09
3	θ_3	90	- 20.32	0
4	θ_4	- 90	0	433.07
5	θ_5	90	0	0
6	θ_6	0	0	56.25

Table 4.1 : paramètres D-H d'un robot PUMA[1]

[1] H), C.Y. Robot Kinematics

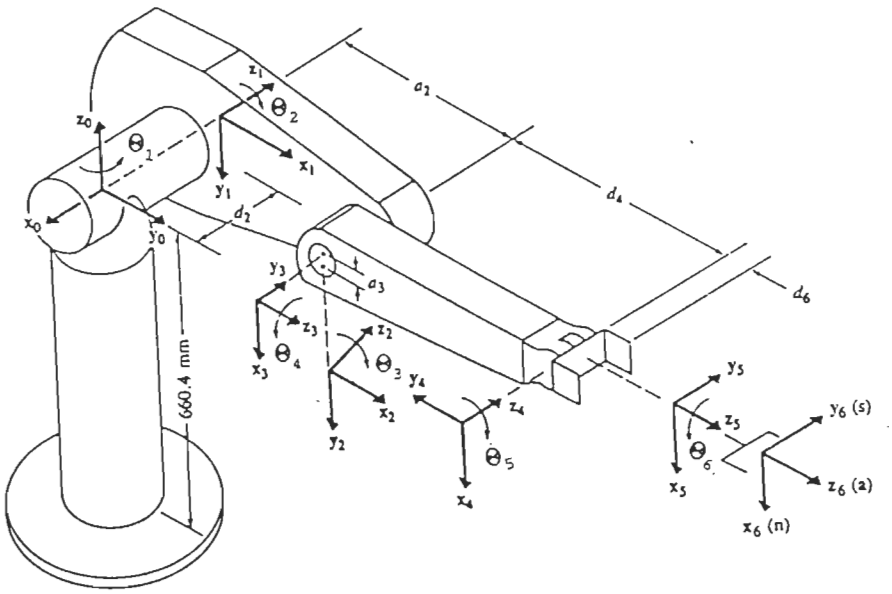


Figure 4.1 : Illustration d'un robot PUMA[1]

La position et l'orientation de départ du bras sont fixées par les valeurs angulaires suivantes :

$\Theta_1 = 80.78°$ $\Theta_2 = 145.75°$ $\Theta_3 = 174.126°$

$\Theta_4 = 122.134°$ $\Theta_5 = 71.483°$ $\Theta_6 = -68.346°$

Ces valeurs définissent la position et l'orientation de départ du bras. Nous simulons alors le travail de l'interpolateur en établissant une trajectoire découpée en jalons égaux.

La trajectoire est un arc de cercle dans un plan parallèle à x_o0y_o, de rayon R = 250mm et centré en un point C tel que C = (-200, -416.2983, 730.81) Nous définissons le long de cet arc de cercle, dix points équidistants tels que chaque jalon mesure environ 8mm. L'orientation du poignet est assez arbitraire; sa variation d'un jalon à l'autre demeure cependant faible (quelques degrés).

L'analyse du passage du bras le long de cet arc de cercle est faite pour deux pas articulaires différents; nous pouvons ainsi observer l'influence du pas articulaire sur la vitesse du bras ainsi que sur la précision de la trajectoire.

a) dans le premier cas, le pas angulaire est STEP = 0.000175 rad soit environ 0.01 degré. Le critère d'arrêt du positionnement est:

$d_i < 0.075$mm

b) dans le deuxième cas, le pas angulaire est STEP = 0.000525 rad soit environ 0.03 degré. Le critère d'arrêt du positionnement est:

$d_i < 0.2$mm

Pour les deux cas, le critère d'arrêt de l'orientation est

$$\beta < 0.00150 \; rad \approx 0.086 \; degré$$

Les résultats permettent également d'analyser l'erreur de position de l'outil que nous noterons Ep_i lors du passage par chaque point avec

$$Ep_i = (dx_i^2 + dy_i^2 + dz_i^2)^{1/2}$$

$$dx_i = x_{ifinal} - x_{ivisé}$$
$$dy_i = y_{ifinal} - y_{ivisé}$$
$$dz_i = z_{ifinal} - z_{ivisé}$$

Les résultats sont présentés sous la forme :

$$x_a \quad y_a \quad z_a \qquad x_w \quad y_w \quad z_w$$

où (x_a, y_a, z_a) désignent les coordonnées de l'extrémité du bras et (x_w, y_w, z_w) désignent les coordonnées du centre de l'outil. Ces résultats sont donnés en millimètres et le nombre d'itérations correspond au nombre de déplacements élémentaires nécessaires pour parcourir le jalon et orienter le poignet.

Lors des simulations, la modification du pas articulaire peut entraîner une modification du critère d'arrêt, car ce dernier dépend à la fois de la tolérance fixée et du pas lui-même. L'algorithme devient divergent dès que le pas articulaire est trop grand pour le critère d'arrêt défini. Dans cette simulation, les valeurs des variables articulaires sont notées après chaque jalon, puis réintroduites manuellement dans le programme pour le jalon suivant, ce qui génère une légère différence entre le point final d'une étape et le point initial de l'autre.

4.2 Résultats

4.2.1 Premier cas : STEP = 0.000175 rad

```
 Coordonnees du point actuel:
-251.559   -619.242   735.433     -301.463   -644.783   730.81

 Coordonnees du point vise:
-243.523   -622.645   735.433     -293.427   -648.185   730.81

 Coordonnees du point final:
-243.556   -622.697   735.403     -293.43   -648.3   730.804

 Nombre d'iterations : 121     Combinaison selectionnee : 175
```

```
 Coordonnees du point actuel:
-243.554   -622.697   735.406     -293.428   -648.299   730.808

 Coordonnees du point vise:
-235.374   -625.764   735.433     -285.278   -651.304   730.81

 Coordonnees du point final:
-235.353   -625.785   735.458     -285.225   -651.393   730.866

 Nombre d'iterations : 128     Combinaison selectionnee : 175
```

Coordonnees du point actuel:
-235.356 -625.785 735.456 -285.228 -651.393 730.864

Coordonnees du point vise:
-227.186 -628.559 736.377 -277.024 -654.137 730.81

Coordonnees du point final:
-227.208 -628.578 736.358 -276.994 -654.172 730.847

Nombre d'iterations : 215 Combinaison selectionnee : 202

Coordonnees du point actuel:
-227.208 -628.579 736.354 -276.996 -654.169 730.846

Coordonnees du point vise:
-218.839 -631.195 736.377 -268.677 -656.68 730.81

Coordonnees du point final:
-218.79 -631.197 736.373 -268.607 -656.73 730.862

Nombre d'iterations : 135 Combinaison selectionnee : 175

Coordonnees du point actuel:
-218.79 -631.199 736.368 -268.607 -656.732 730.857

Coordonnees du point vise:
-210.408 -633.449 736.377 -260.246 -658.931 730.81

Coordonnees du point final:
-210.435 -633.471 736.364 -260.257 -658.995 730.858

Nombre d'iterations : 130 Combinaison selectionnee : 202

```
 Coordonnees du point actuel:
-210.438   -633.469   736.362      -260.26   -658.992   730.856

 Coordonnees du point vise:
-202.016   -635.404   736.596      -251.741   -660.885   730.81

 Coordonnees du point final:
-202.046   -635.45   736.626      -251.835   -660.98   730.855

 Nombre d'iterations : 152      Combinaison selectionnee : 202


 Coordonnees du point actuel:
-202.049   -635.449   736.626      -251.838   -660.979   730.855

 Coordonnees du point vise:
-193.449   -637.061   736.596      -243.174   -662.542   730.81

 Coordonnees du point final:
-193.511   -637.083   736.628      -243.292   -662.626   730.846

 Nombre d'iterations : 130      Combinaison selectionnee : 175


 Coordonnees du point actuel:
-193.511   -637.084   736.625      -243.292   -662.627   730.843

 Coordonnees du point vise:
-184.829   -638.418   736.596      -234.554   -663.899   730.81

 Coordonnees du point final:
-184.842   -638.437   736.604      -234.622   -663.98   730.803

 Nombre d'iterations : 133      Combinaison selectionnee : 175
```

```
 Coordonnees du point actuel:
-184.846   -638.436   736.601      -234.625   -663.979   730.8

 Coordonnees du point vise:
-176.23    -639.529   737.927      -225.899   -664.954   730.81

 Coordonnees du point final:
-176.285   -639.575   737.948      -225.939   -665.047   730.894

 Nombre d'iterations : 247     Combinaison selectionnee : 302
```

```
 Coordonnees du point actuel:
-176.285   -639.575   737.948      -225.939   -665.047   730.894

 Coordonnees du point vise:
-167.529   -640.281   737.927      -217.198   -665.706   730.81

 Coordonnees du point final:
-167.493   -640.321   737.877      -217.127   -665.819   730.784

 Nombre d'iterations : 133     Combinaison selectionnee : 175
```

4.2.2 Deuxième cas : STEP = 0.000525 rad

```
 Coordonnees du point actuel:
-251.559   -619.242   735.433     -301.463   -644.783   730.81

 Coordonnees du point vise:
-243.523   -622.645   735.433     -293.427   -648.185   730.81

 Coordonnees du point final:
-243.454   -622.782   735.466     -293.335   -648.365   730.837

 Nombre d'iterations : 41     Combinaison selectionnee : 175


 Coordonnees du point actuel:
-243.454   -622.783   735.464     -293.326   -648.385   730.835

 Coordonnees du point vise:
-235.374   -625.764   735.433     -285.278   -651.304   730.81

 Coordonnees du point final:
-235.16   -625.924   735.464     -285.14   -651.504   730.809

 Nombre d'iterations : 44     Combinaison selectionnee : 94


 Coordonnees du point actuel:
-235.363   -625.923   735.464     -285.143   -651.503   730.809

 Coordonnees du point vise:
-227.186   -628.559   736.377     -277.024   -654.137   730.81

 Coordonnees du point final:
-227.316   -628.593   736.303     -277.12   -654.176   730.8

 Nombre d'iterations : 88     Combinaison selectionnee : 202
```

Coordonnees du point actuel:
-227.338 -628.595 736.301 -277.121 -654.177 730.798

Coordonnees du point vise:
-218.839 -631.195 736.377 -268.677 -656.68 730.81

Coordonnees du point final:
-218.677 -631.18 736.423 -268.491 -656.715 730.898

Nombre d'iterations : 46 Combinaison selectionnee : 175

Coordonnees du point actuel:
-218.677 -631.182 736.418 -268.491 -656.717 730.893

Coordonnees du point vise:
-210.408 -633.449 736.377 -260.246 -658.931 730.81

Coordonnees du point final:
-210.318 -633.411 736.352 -260.127 -658.948 730.786

Nombre d'iterations : 43 Combinaison selectionnee : 175

Coordonnees du point actuel:
-210.321 -633.41 736.352 -260.13 -658.946 730.786

Coordonnees du point vise:
-202.016 -635.404 736.596 -251.741 -660.885 730.81

Coordonnees du point final:
-201.938 -635.487 736.626 -251.726 -661.013 730.828

Nombre d'iterations : 50 Combinaison selectionnee : 202

```
 Coordonnees du point actuel:
-201.941   -635.486   736.623      -151.73   -661.012   730.825

 Coordonnees du point vise:
-193.449   -637.061   736.596      -243.174  -662.542   730.81

 Coordonnees du point final:
-193.522   -637.197   736.523      -243.307  -662.725   730.703

 Nombre d'iterations : 44      Combinaison selectionnee : 94


 Coordonnees du point actuel:
-193.525   -637.197   736.519      -243.31   -662.724   730.7

 Coordonnees du point vise:
-184.829   -638.418   736.596      -234.554  -663.899   730.81

 Coordonnees du point final:
-184.735   -638.427   736.539      -234.521  -663.953   730.721

 Nombre d'iterations : 46      Combinaison selectionnee : 94


 Coordonnees du point actuel:
-184.735   -638.426   736.54       -234.521  -663.951   730.723

 Coordonnees du point vise:
-176.23    -639.529   737.927      -225.899  -664.954   730.81

 Coordonnees du point final:
-176.288   -639.576   737.95       -225.951  -665.026   730.887

 Nombre d'iterations : 91      Combinaison selectionnee : 202
```

```
 Coordonnees du point actuel:
-176.291   -639.573   737.948     -225.954   -665.024   730.885

 Coordonnees du point vise:
-167.529   -640.281   737.927     -217.198   -665.706   730.81

 Coordonnees du point final:
-167.497   -640.263   737.927     -217.151   -665.732   730.863

 Nombre d'iterations : 45     Combinaison selectionnee : 202
```

	STEP = 0.000175	STEP = 0.000525
Ep_1	0.1152	0.2261
Ep_2	0.1177	0.243
Ep_3	0.0591	0.1041
Ep_4	0.0938	0.2087
Ep_5	0.0808	0.1226
Ep_6	0.1410	0.1301
Ep_7	0.1492	0.250
Ep_8	0.106	0.1092
Ep_9	0.1315	0.1175
Ep_{10}	0.136	0.0754

Table 4.2 : Erreurs de position en mm

D'après ces résultats, on peut observer qu'en multipliant le pas articulaire par trois,on réduit d'environ 1/3 le nombre d'itérations nécessaires pour parcourir le même jalon.

Cela implique une réduction appréciable du temps d'exécution de la tâche, et ce d'autant plus que l'erreur de position Ep_i ne subit pas une trop grande variation lors du passage par les points spécifiés. Il faut noter que dans le premier cas, le déplacement élémentaire maximum est inférieur à 0.2 mm et dans le second cas à 0.3 mm.

CONCLUSION

Ce projet a été réalisé dans le but d'élaborer une nouvelle méthode de génération des trajectoire pour l'activation des bras de robot; pour cela nous avons admis comme hypothèse de départ que la technologie actuelle permet la fabrication de servomécanismes à réponse rapide, stables et bien amortis, ainsi que celle de codeurs optiques capables de déceler de très faibles variations angulaires ou linéaires. Dès lors, il devient possible d'adopter une méthode itérative comme celle décrite dans ce travail.

Dans tous les ouvrages consultés au cours de la recherche bibliographique, la génération des trajectoires est toujours associée à des algorithmes de cinématique inverse; aussi pouvons-nous souligner, avec une certaine satisfaction, l'innovation qu'apporte la méthode présentée.

1. Caractéristiques principales

L'élaboration de cette nouvelle méthode ne s'est pas faite sans difficultés, le point le plus épineux étant la détermination de tous les paramètres dynamiques et particulièrement la mise en évidence des paramètres à contrôler. Néanmoins, après l'écriture des programmes et la simulation, les résultats obtenus se sont

avérés fort encourageants.

Le principe de génération de la trajectoire, basé sur la cinématique directe associée aux petites variations, aboutit à un algorithme dont la simplicité facilite l'optimisation des supports matériel et logiciel permettant de l'exploiter. La matrice jacobienne évolue très lentement et peut par conséquent n'être réévaluée qu'après un certain nombre d'itérations; cela simplifie davantage des opérations déjà peu complexes, réduit considérablement le temps de calcul et facilite l'utilisation de processeurs peu coûteux.

Par ailleurs, l'approche itérative utilisée élimine les solutions ambiguës généralement introduites par les algorithmes de cinématique inverse: les pas articulaires étant très petits, les configurations singulières ne posent pas un problème de choix. La solution au problème cinématique est toujours unique et la poursuite de la trajectoire est autocorrective; en effet, lorsque des points de la trajectoire ont été spécifiés, le bras ne peut pas s'écarter de celle-ci car, comme nous pouvons le voir dans la table 4.2, les erreurs de position ne sont pas cumulatives.

Cependant, bien que les résultats obtenus concordent avec les objectifs que nous nous étions fixés, cette nouvelle méthode nécessite une étude plus élaborée afin d'en exploiter judicieusement les performances.

2. Développements futurs

Un problème partiellement abordé dans cette étude est le contrôle de la vitesse moyenne de déplacement du bras. Pour avoir une bonne idée des vitesses pouvant être atteintes et de la stratégie de contrôle la plus appropriée, il est indispensable d'appliquer la méthode à un système réel ou réalisable. Il s'agit par conséquent d'utiliser des processeurs bien précis ainsi que toutes les caractéristiques dynamiques réelles d'un système robotisé afin d'évaluer le temps d'exécution d'une itération; ceci détermine la période minimale d'échantillonnage des commandes du processeur principal et permet de calculer la vitesse maximale théorique. Le problème du contôle de la vitesse pourrait par conséquent faire l'objet d'une étude plus approfondie dans le cadre d'un autre projet.

D'autre part, même si l'on admet aisément que le niveau technologique actuel permet de réaliser un système robotisé répondant à nos exigences, il est indispensable de vérifier, en pratique, le comportement dynamique d'un bras activé selon la méthode développée; cette étude permettrait de définir la ou les catégories de robot ainsi que les types de tâches pour lesquels la méthode offre les meilleures performances d'une part, et, d'autre part d'établir, si besoin est, les limites actuelles des servomécanismes utilisés en robotique.

Quoiqu'il en soit, le but de ce travail n'était pas de mettre au point un produit directement utilisable sur le marché: cela n'aurait pas été possible compte

tenu de nos moyens logiciels et matériels limités; ni de faire une comparaison immédiate avec des méthodes ou des logiciels existant déjà. Cette comparaison pourra cependant être possible à l'issue de développements futurs apportés à la méthode présentée.

Nous avons voulu explorer un aspect de la commande en robotique toujours mis à l'écart par les chercheurs, et qui pourtant semblait offrir une résolution plus simple du problème cinématique, soit en d'autres termes développer un nouveau produit susceptible d'être efficace.

Les résultats de notre recherche nous permettent d'affirmer que l'activation des bras manipulateurs peut se faire autrement que par des algorithmes complexes de cinématique inverse, et ce, tout en conservant une bonne précision dans la poursuite de la trajectoire. Par la simplicité de ses algorithmes, cette nouvelle méthode semble offir des perspectives intéressantes pour la commande des systèmes robotisés; aussi espérons-nous qu'elle pourra être approfondie davantage dans un futur proche.

BIBLIOGRAPHIE

Pierre,ANDRÉ, J-M,KAUFFMANN, F.,LHOTE, J-P,TAILLARD. *Les Robots: constituants technologiques*, Paris, Hermès Publishing, 1983, Tome 4.

Chae H.,ANN, Christopher G.,ATEKSON, John M.,HOLLERBACH. *Model-based control of a robot manipulator*, Massachusetts Institute of Technology, The MIT Press, 1988.

H.,ASADA and J-J.E.,SLOTINE. *Robot analysis and control*, Massachusetts Institute of Technology, Wiley-Interscience Publication, 1986.

Philippe,COIFFET. *Les Robots: modélisation et commande*, Neully, Hermès Publishing, 1981, Tome1.

C.Y.,HO and Jen,SRIWATTANATHAMMA. *Robots kinematics: symbolic automation and numerical synthesis*, New-Jersey, Ablex Publishing Corporation, 1990.

G.,LACROUX. *Les actionneurs électriques pour la robotique et les asservissements*, Paris, Technique et Documentation(Lavoisier), 1985.

Louis,MARET. *Régulation automatique*, Lausanne, Presses Polytechniques Romandes, 1987.

Miomir,VUKOBRATOVIC. *Introduction to robotics*, Springler-Verlag, 1989.

ANNEXE A

Programme de validation de la méthode

```
float gx(float *t,float *a);
float hx(float *t,float *a);
float fy(float *t,float *a);
float gy(float *t,float *a);
float hy(float *t,float *a);
float fz(float *t,float *a);
float gz(float *t,float *a);
float hz(float *t,float *a);
float ft6(float *t,float *a);
float gt6(float *t,float *a);
float ht6(float *t,float *a);

int rep,m;
int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
float *t,*a,*al,*d,*Tt,*Ttmax,STEP,*pas_bal;
float *J[3];
int *Ct,*Cr;
float **Q,*Tab;
float stepR,stepT;

main()
{
printf("INTRODUCTION DES PARAMETRES CONSTANTS\n");

/* Il s'agit d'introduire le nombre d'articulations du bras, les parametres*/
/* du bras,les intervalles de variation de chaque articulation, le pas de */
/* balayage et la precision desiree.*/

printf("entrer le nombre d'articulations n:  ");fflush(stdout);

scanf("%d",&n);
Nb_comb=(pow(3,n));
printf("\n   Nombre d'articulations : %d\n\n",n);
printf("   Nombre de combinaisons : %d\n\n",Nb_comb);
printf("   Y a-t-il des articulations qui sont des translations?\n");
printf("\n   Si oui taper 1, si non taper 0:  ");fflush(stdout);
scanf("%d",&rep);
switch(rep)
   {
   case 1: printf("\n  Entrer le nombre de translations :  ");fflush(stdout);
           scanf("%d",&Nb_trans);
           printf("\n  Entrer le pas lineaire: stepT  ");fflush(stdout);
           scanf("%g",&stepT);
           printf("\n  Entrer le pas angulaire: stepR  ");fflush(stdout);
           scanf("%g",&stepR);
           break;
   case 0: Nb_trans=0;
            printf("\n  Entrer le pas angulaire: stepR   ");fflush(stdout);
            scanf("%g",&stepR);
   }

Trans=Calloc(Nb_trans,int);           /*Vecteur des translations*/

if (Nb_trans !=0)

   for (i=0; i<Nb_trans; i++)
     {
     printf("\n  Entrer le numero de la translation :  ");fflush(stdout);
     scanf("%d",&Trans[i]);
     }
```

```
/*   INTRODUCTION DES PARAMETRES DE D-H    */
/* Le vecteur "T" correspond aux angles theta; le vecteur "a" aux angles alpha*/
/* le vecteur "al" aux constantes ai et le vecteur "d" aux constantes di . */

t=Calloc(n,TRUC);
a=Calloc(n,TRUC);
al=Calloc(n,TRUC);
d=Calloc(n,TRUC);

printf("\n  Introduire les parametres constants et initialiser les \n");
printf("  parametres variables a zero.\n");fflush(stdout);

printf("\n        INTRODUCTION DES PARAMETRES THETA   \n");
for(i=0; i<n; i++)
   {
     printf("  theta[ %d ]=\t",i);fflush(stdout);
     scanf("%g",&t[i]);
   }
printf("\n        INTRODUCTION DES PARAMETRES ALPHA    \n");
for(i=0; i<n; i++)
   {
     printf("  alpha[ %d ]=\t",i);fflush(stdout);
     scanf("%g",&a[i]);
   }
printf("\n         INTRODUCTION DES CONSTANTES ai      \n");
for(i=0; i<n; i++)
   {
     printf("  a[ %d ]=\t",i);fflush(stdout);
     scanf("%g",&al[i]);
   }
printf("\n         INTRODUCTION DES PARAMETRES di       \n");
for (i=0; i<n; i++)
  {
    printf("  d[ %d ]=\t",i);fflush(stdout);
    scanf("%g",&d[i]);
  }

  Tt=Calloc(n,TRUC); /* Tt est le vecteur des variables articulaires,
                        comprenant rotations et translations. */

Ttmax=Calloc(n,TRUC); /* Ttmax est le vecteur des valeurs maximales de
                         chaque variable articulaire. */

pas_bal=Calloc(n,TRUC); /* Vecteur contenant le pas de balayage pour
                           chaque articulation. */

Ct=Calloc(n,int);
Cr=Calloc(n,int);



 for (i=0; i<n; i++)
    {
    printf("\n  Max articulation( %d )=\t",i);fflush(stdout);
    scanf("%g",&Ttmax[i]);
    }
```

```
for (i=0; i<n; i++)
   {
   printf("\n  Pas de balayage articulation( %d ):\t",i);fflush(stdout);
   scanf("%g",&pas_bal[i]);
   }

Tab=Calloc((n*Nb_comb),TRUC);
Q=Calloc(Nb_comb,TRUC *);
if (Tab==NULL || Q==NULL) exit(1);
for (i=0; i<Nb_comb; i++)
   {
   Q[i]=Tab+i*n;
   }

/*  Allocation d'espace pour la matrice jacobienne */

 for (i=0; i<3; i++)
   {
    J[i]=Calloc(n,TRUC);
   }


   matq();

   switch(n)
    {
      case 3:  t[3]=0;  d[3]=0;
               t[4]=0;  d[4]=0;
               t[5]=0;  d[5]=0;
               jacobi3();
               break;
       case 4: t[4]=0;  d[4]=0;
               t[5]=0;  d[5]=0;
               jacobi4();
               break;
       case 5: t[5]=0;  d[5]=0;
               jacobi5();
               break;
        case 6: jacobi6();

       }
  }
```

```
/*  ========================================================================
    *             DEFINITION DES FONCTIONS DEVANT SERVIR A ETABLIR          *
    *                        LA MATRICE JACOBIENNE                          *
    ======================================================================== */


#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)


float fx(t,a)
float *t,*a;
{
   return((C(t[0])*C(t[1])*C(t[2]))-(S(t[0])*C(a[0])*S(t[1])*C(t[2]))
          -(C(t[0])*S(t[1])*C(a[1])*S(t[2]))-(S(t[0])*C(a[0])*C(t[1])
          *C(a[1])*S(t[2]))+(S(t[0])*S(a[0])*S(a[1])*S(t[2])));
}

float gx(t,a)
float *t,*a;
{
  return((-C(t[0])*C(t[1])*S(t[2])*C(a[2]))+(S(t[0])*C(a[0])*S(t[1])
         *S(t[2])*C(a[2]))-(C(t[0])*S(t[1])*C(a[1])*C(t[2])*C(a[2]))
         -(S(t[0])*C(a[0])*C(t[1])*C(a[1])*C(t[2])*C(a[2]))+(S(t[0])
         *S(a[0])*S(a[1])*C(t[2])*C(a[2]))+(C(t[0])*S(t[1])*S(a[1])
         *S(a[2]))+(S(t[0])*C(a[0])*C(t[1])*S(a[1])*S(a[2]))+(S(t[0])
         *S(a[0])*C(a[1])*S(a[2])));
}

float hx(t,a)
float *t,*a;
{
  return((C(t[0])*C(t[1])*S(t[2])*S(a[2]))-(S(t[0])*C(a[0])*S(t[1])
         *S(t[2])*S(a[2]))+(C(t[0])*S(t[1])*C(a[1])*C(t[2])*S(a[2]))
         +(S(t[0])*C(a[0])*C(t[1])*C(a[1])*C(t[2])*S(a[2]))-(S(t[0])
         *S(a[0])*S(a[1])*C(t[2])*S(a[2]))+(C(t[0])*S(t[1])*S(a[1])
         *C(a[2]))+(S(t[0])*C(a[0])*C(t[1])*S(a[1])*C(a[2]))+(S(t[0])
         *S(a[0])*C(a[1])*C(a[2])));
}

float fy(t,a)
float *t,*a;
{
  return((S(t[0])*C(t[1])*C(t[2]))+(C(t[0])*C(a[0])*S(t[1])*C(t[2]))
         -(S(t[0])*S(t[1])*C(a[1])*S(t[2]))+(C(t[0])*C(a[0])*C(t[1])
         *C(a[1])*S(t[2]))-(C(t[0])*S(a[0])*S(a[1])*S(t[2])));
}
```

```
float gy(t,a)
float *t,*a;
{
  return((-S(t[0])*C(t[1])*S(t[2])*C(a[2]))-(C(t[0])*C(a[0])*S(t[1])
         *S(t[2])*C(a[2]))-(S(t[0])*S(t[1])*C(a[1])*C(t[2])*C(a[2]))
         +(C(t[0])*C(a[0])*C(t[1])*C(a[1])*C(t[2])*C(a[2]))-(C(t[0])
         *S(a[0])*S(a[1])*C(t[2])*C(a[2]))+(S(t[0])*S(t[1])*S(a[1])
         *S(a[2]))-(C(t[0])*C(a[0])*C(t[1])*S(a[1])*S(a[2]))-(C(t[0])
         *C(a[0])*C(a[1])*S(a[2])));
}

float hy(t,a)
float *t,*a;
{
  return((S(t[0])*C(t[1])*S(t[2])*S(a[2]))+(C(t[0])*C(a[0])*S(t[1])
         *S(t[2])*S(a[2]))+(S(t[0])*S(t[1])*C(a[1])*C(t[2])*S(a[2]))
         -(C(t[0])*C(a[0])*C(t[1])*C(a[1])*C(t[2])*S(a[2]))+(C(t[0])
         *S(a[0])*S(a[1])*C(t[2])*S(a[2]))+(S(t[0])*S(t[1])*S(a[1])
         *C(a[2]))-(C(t[0])*C(a[0])*C(t[1])*S(a[1])*C(a[2]))-(C(t[0])
         *S(t[0])*S(a[1])*C(a[2])));
}

float fz(t,a)
float *t,*a;
{
  return((S(a[0])*S(t[1])*C(t[2]))+(S(a[0])*C(t[1])*C(a[1])*S(t[2]))
         +(C(a[0])*S(a[1])*S(t[2])));
}

float gz(t,a)
float *t,*a;
{
  return((-S(a[0])*S(t[1])*S(t[2])*C(a[2]))+(S(a[0])*C(t[1])*C(a[1])
         *C(t[2])*C(a[2]))+(C(a[0])*S(a[1])*C(t[2])*C(a[2]))-(S(a[0])
         *C(t[1])*S(a[1])*S(a[2]))+(C(a[0])*C(a[1])*S(a[2])));
}

float hz(t,a)
float *t,*a;
{
  return((S(a[0])*S(t[1])*S(t[2])*S(a[2]))-(S(a[0])*C(t[1])*C(a[1])
         *C(t[2])*S(a[2]))-(C(a[0])*S(a[1])*C(t[2])*S(a[2]))-(S(a[0])
         *C(t[1])*S(a[1])*C(a[2]))+(C(a[0])*C(a[1])*C(a[2])));
}

float fp(t,a,d,al)
float *t,*a,*al,*d;

{
 return((d[5]*((C(t[3])*S(t[4])*S(a[4]))+(S(t[3])*C(a[3])*C(t[4])
        *S(a[4]))+(S(t[3])*S(a[3])*C(a[4]))))+(al[4]*((C(t[3])
        *C(t[4]))-(S(t[3])*C(a[3])*S(t[4]))))+(d[4]*S(t[3])
        *S(a[3]))+(al[3]*C(t[3])));
}
```

```
float gp(t,a,d,al)
float *t,*a,*d,*al;
{
 return((d[5]*((S(t[3])*S(t[4])*S(a[4]))-(C(t[3])*C(a[3])*C(t[4])
        *S(a[4]))-(C(t[3])*S(a[3])*C(a[4]))))+(al[4]*((S(t[3])
        *C(t[4]))+(C(t[3])*C(a[3])*S(t[4]))))-(d[4]*C(t[3])
        *S(a[3]))+(al[3]*S(t[3])));
}

float hp(t,a,d,al)
float *t,*a,*d,*al;
{
 return((-d[5]*((S(a[3])*C(t[4])*S(a[4]))+(C(a[3])*C(a[4]))))
        +(al[4]*S(a[3])*S(t[4]))-(d[4]*C(a[3]))+d[3]);
}

float dfx1(t,a)
float *t,*a;
{
   return((-S(t[0])*C(t[1])*C(t[2]))-(C(t[0])*C(a[0])*S(t[1])
          *C(t[2]))+(S(t[0])*S(t[1])*C(a[1])*S(t[2]))-(C(t[0])
          *C(a[0])*C(t[1])*C(a[1])*S(t[2]))+(C(t[0])*S(a[0])*S(a[1])*S(t[2])));
}

float dgx1(t,a)
float *t,*a;
{
  return((S(t[0])*C(t[1])*S(t[2])*C(a[2]))+(C(t[0])*C(a[0])*S(t[1])
         *S(t[2])*C(a[2]))+(S(t[0])*S(t[1])*C(a[1])*C(t[2])*C(a[2]))
         -(C(t[0])*C(a[0])*C(t[1])*C(a[1])*C(t[2])*C(a[2]))+(C(t[0])
         *S(a[0])*S(a[1])*C(t[2])*C(a[2]))-(S(t[0])*S(t[1])*S(a[1])
         *S(a[2]))+(C(t[0])*C(a[0])*C(t[1])*S(a[1])*S(a[2]))+(C(t[0])
         *S(a[0])*C(a[1])*S(a[2])));
}

float dhx1(t,a)
float *t,*a;
{
  return((-S(t[0])*C(t[1])*S(t[2])*S(a[2]))-(C(t[0])*C(a[0])*S(t[1])
         *S(t[2])*S(a[2]))-(S(t[0])*S(t[1])*C(a[1])*C(t[2])*S(a[2]))
         +(C(t[0])*C(a[0])*C(t[1])*C(a[1])*C(t[2])*S(a[2]))-(C(t[0])
         *S(a[0])*S(a[1])*C(t[2])*S(a[2]))-(S(t[0])*S(t[1])*S(a[1])
         *C(a[2]))+(C(t[0])*C(a[0])*C(t[1])*S(a[1])*C(a[2]))+(C(t[0])
         *S(a[0])*C(a[1])*C(a[2])));
}

float dx1(t,a,d,al)
float *t,*a,*d,*al;
{
 return((al[2]*((-S(t[0])*C(t[1])*C(t[2]))-(C(t[0])*C(a[0])*S(t[1])
        *S(t[2]))+(S(t[0])*S(t[1])*C(a[1])*S(t[2]))-(C(t[0])*C(a[0])
        *C(t[1])*C(a[1])*S(t[2]))+(C(t[0])*S(a[0])*S(a[1])*S(t[2]))))
        +(d[2]*((-S(t[0])*S(t[1])*S(a[1]))+(C(t[0])*C(a[0])*C(t[1])
        *S(a[1]))+(C(t[0])*S(a[0])*C(a[1]))))+(al[1]*((-S(t[0])
        *C(t[1]))-(C(t[0])*C(a[0])*S(t[1]))))+(d[1]*C(t[0])*S(a[0]))
        -(al[0]*S(t[0])));
}
```

```
float dfx2(t,a)
float *t,*a;
{
  return((-C(t[0])*S(t[1])*C(t[2]))-(S(t[0])*C(a[0])*C(t[1])*C(t[2]))
         -(C(t[0])*C(t[1])*C(a[1])*S(t[2]))+(S(t[0])*C(a[0])*S(t[1])
         *C(a[1])*S(t[2])));
}

float dgx2(t,a)
float *t,*a;
{
  return((C(t[0])*S(t[1])*S(t[2])*C(a[2]))+(S(t[0])*C(a[0])*C(t[1])
         *S(t[2])*C(a[2]))-(C(t[0])*C(t[1])*C(a[1])*C(t[2])*C(a[2]))
         +(S(t[0])*C(a[0])*S(t[1])*C(a[1])*C(t[2])*C(a[2]))+(C(t[0])
         *C(t[1])*S(a[1])*S(a[2]))-(S(t[0])*C(a[0])*S(t[1])*S(a[1])*S(a[2])));
}

float dhx2(t,a)
float *t,*a;
{
  return((-C(t[0])*S(t[1])*S(t[2])*S(a[2]))-(S(t[0])*C(a[0])*C(t[1])
         *S(t[2])*S(a[2]))+(C(t[0])*C(t[1])*C(a[1])*C(t[2])*S(a[2]))
         -(S(t[0])*C(a[0])*S(t[1])*C(a[1])*C(t[2])*S(a[2]))+(C(t[0])
         *C(t[1])*S(a[1])*C(a[2]))-(S(t[0])*C(a[0])*S(t[1])*S(a[1])*C(a[2])));
}

float dx2(t,a,d,al)
float *t,*a,*al,*d;
{
  return((al[2]*((-C(t[0])*S(t[1])*C(t[2]))-(S(t[0])*C(a[0])*C(t[1])
         *C(t[2]))-(C(t[0])*C(t[1])*C(a[1])*S(t[2]))÷(S(t[0])*C(a[0])
         *S(t[1])*C(a[1])*S(t[2]))))+(d[2]*((C(t[0])*C(t[1])*S(a[1]))
         -(S(t[0])*C(a[0])*S(t[1])*S(a[1]))))÷(al[1]*((-C(t[0])*S(t[1]))
         -(S(t[0])*C(a[0])*C(t[1])))));
}

float dfx3(t,a)
float *t,*a;
{
  return((-C(t[0])*C(t[1])*S(t[2]))+(S(t[0])*C(a[0])*S(t[1])*S(t[2]))
         -(C(t[0])*S(t[1])*C(a[1])*C(t[2]))-(S(t[0])*C(a[0])*C(t[1])
         *C(a[1])*C(t[2]))+(S(t[0])*S(a[0])*S(a[1])*C(t[2])));
}

float dgx3(t,a)
float *t,*a;
{
  return((-C(t[0])*C(t[1])*C(t[2])*C(a[2]))+(S(t[0])*C(a[0])*S(t[1])
         *C(t[2])*C(a[2]))+(C(t[0])*S(t[1])*C(a[1])*S(t[2])*C(a[2]))
         +(S(t[0])*C(a[0])*C(t[1])*C(a[1])*S(t[2])*C(a[2]))-(S(t[0])
         *S(a[0])*S(a[1])*S(t[2])*C(a[2])));
}
```

```
float dhx3(t,a)
float *t,*a;
{
  return((C(t[0])*C(t[1])*C(t[2])*S(a[2]))-(S(t[0])*C(a[0])*S(t[1])
         *C(t[2])*S(a[2]))-(C(t[0])*S(t[1])*C(a[1])*S(t[2])*S(a[2]))
         -(S(t[0])*C(a[0])*C(t[1])*C(a[1])*S(t[2])*S(a[2]))+(S(t[0])
         *S(a[0])*S(a[1])*S(t[2])*S(a[2])));
}

float dx3(t,a,al)
float *t,*a,*al;
{
  return(al[2]*((-C(t[0])*C(t[1])*S(t[2]))+(S(t[0])*C(a[0])*S(t[1])
         *S(t[2]))-(C(t[0])*S(t[1])*C(a[1])*C(t[2]))-(S(t[0])*C(a[0])
         *C(t[1])*C(a[1])*C(t[2]))+(S(t[0])*S(a[0])*S(a[1])*C(t[2]))));
}

float dfy1(t,a)
float *t,*a;
{
  return((C(t[0])*C(t[1])*C(t[2]))-(S(t[0])*C(a[0])*S(t[1])*C(t[2]))
         -(C(t[0])*S(t[1])*C(a[1])*S(t[2]))-(S(t[0])*C(a[0])*C(t[1])
         *C(a[1])*S(t[2]))+(S(t[0])*S(a[0])*S(a[1])*S(t[2])));
}

float dgy1(t,a)
float *t,*a;
{
  return((-C(t[0])*C(t[1])*S(t[2])*C(a[2]))+(S(t[0])*C(a[0])*S(t[1])
         *S(t[2])*C(a[2]))-(C(t[0])*S(t[1])*C(a[1])*C(t[2])*C(a[2]))
         -(S(t[0])*C(a[0])*C(t[1])*C(a[1])*C(t[2])*C(a[2]))+(S(t[0])
         *S(a[0])*S(a[1])*C(t[2])*C(a[2]))+(C(t[0])*S(t[1])*S(a[1])
         *S(a[2]))+(S(t[0])*C(a[0])*C(t[1])*S(a[1])*S(a[2]))+(S(t[0])
         *C(a[0])*C(a[1])*S(a[2])));
}

float dhy1(t,a)
float *t,*a;
{
  return((C(t[0])*C(t[1])*S(t[2])*S(a[2]))-(S(t[0])*C(a[0])*S(t[1])*S(t[2])
         *S(a[2]))+(C(t[0])*S(t[1])*C(a[1])*C(t[2])*S(a[2]))+(S(t[0])
         *C(a[0])*C(t[1])*C(a[1])*C(t[2])*S(a[2]))-(S(t[0])*S(a[0])
         *S(a[1])*C(t[2])*S(a[2]))+(C(t[0])*S(t[1])*S(a[1])*C(a[2]))
         +(S(t[0])*C(a[0])*C(t[1])*S(a[1])*C(a[2]))+(S(t[0])*S(a[0])
         *C(a[1])*C(a[2])));
}

float dy1(t,a,d,al)
float *t,*a,*al,*d;
{
  return((al[2]*((C(t[0])*C(t[1])*C(t[2]))-(S(t[0])*C(a[0])*S(t[1])
         *C(t[2]))-(C(t[0])*S(t[1])*C(a[1])*S(t[2]))-(S(t[0])*C(a[0])
         *C(t[1])*C(a[1])*S(t[2]))+(S(t[0])*S(a[0])*S(a[1])*S(t[2]))))
         +(d[2]*((C(t[0])*S(t[1])*S(a[1]))+(S(t[0])*C(a[0])*C(t[1])
         *S(a[1]))+(S(t[0])*S(a[0])*C(a[1]))))+(al[1]*((C(t[0])*C(t[1]))
         -(S(t[0])*C(a[0])*S(t[1]))))+(d[1]*S(t[0])*S(a[0]))+(al[0]*C(t[0])));
}
```

```
float dfy2(t,a)
float *t,*a;
{
  return((-S(t[0])*S(t[1])*C(t[2]))+(C(t[0])*C(a[0])*C(t[1])*C(t[2]))
         -(S(t[0])*C(t[1])*C(a[1])*S(t[2]))-(C(t[0])*C(a[0])*S(t[1])
         *C(a[1])*S(t[2])));
}

float dgy2(t,a)
float *t,*a;
{
  return((S(t[0])*S(t[1])*S(t[2])*C(a[2]))-(C(t[0])*C(a[0])*C(t[1])
         *S(t[2])*C(a[2]))-(S(t[0])*C(t[1])*C(a[1])*C(t[2])*C(a[2]))
         -(C(t[0])*C(a[0])*S(t[1])*C(a[1])*C(t[2])*C(a[2]))÷(S(t[0])
         *C(t[1])*S(a[1])*S(a[2]))÷(C(t[0])*C(a[0])*S(t[1])*S(a[1])*S(a[2])));
}

float dhy2(t,a)
float *t,*a;
{
  return((-S(t[0])*S(t[1])*S(t[2])*S(a[2]))÷(C(t[0])*C(a[0])*C(t[1])
         *S(t[2])*S(a[2]))+(S(t[0])*C(t[1])*C(a[1])*C(t[2])*S(a[2]))
         ÷(C(t[0])*C(a[0])*S(t[1])*C(a[1])*C(t[2])*S(a[2]))+(S(t[0])
         *C(t[1])*S(a[1])*C(a[2]))÷(C(t[0])*C(a[0])*S(t[1])*S(a[1])*C(a[2])));
}

float dy2(t,a,d,al)
float *t,*a,*al,*d;
{
  return((al[2]*((-S(t[0])*S(t[1])*C(t[2]))÷(C(t[0])*C(a[0])*C(t[1])
         *C(t[2]))-(S(t[0])*C(t[1])*C(a[1])*S(t[2]))-(C(t[0])*C(a[0])
         *S(t[1])*C(a[1])*S(t[2]))))÷(d[2]*((S(t[0])*C(t[1])*S(a[1]))
         ÷(C(t[0])*C(a[0])*S(t[1])*S(a[1]))))÷(al[1]*((-S(t[0])*S(t[1]))
         ÷(C(t[0])*C(a[0])*C(t[1])))));
}

float dfy3(t,a)
float *t,*a;
{
  return((-S(t[0])*C(t[1])*S(t[2]))-(C(t[0])*C(a[0])*S(t[1])*S(t[2]))
         -(S(t[0])*S(t[1])*C(a[1])*C(t[2]))÷(C(t[0])*C(a[0])*C(t[1])
         *C(t[2]))-(C(t[0])*S(a[0])*S(a[1])*C(t[2])));
}

float dgy3(t,a)
float *t,*a;
{
  return((-S(t[0])*C(t[1])*C(t[2])*C(a[2]))-(C(t[0])*C(a[0])*S(t[1])
         *C(t[2])*C(a[2]))+(S(t[0])*S(t[1])*C(a[1])*S(t[2])*C(a[2]))
         -(C(t[0])*C(a[0])*C(t[1])*C(a[1])*S(t[2])*C(a[2]))-(C(t[0])
         *S(a[0])*S(a[1])*S(t[2])*C(a[2])));
}
```

```
float dhy3(t,a)
float *t,*a;
{
  return((S(t[0])*C(t[1])*C(t[2])*S(a[2]))+(C(t[0])*C(a[0])*S(t[1])
         *C(t[2])*S(a[2]))-(S(t[0])*S(t[1])*C(a[1])*S(t[2])*S(a[2]))
         +(C(t[0])*C(a[0])*C(t[1])*C(a[1])*S(t[2])*S(a[2]))-(C(t[0])
         *S(a[0])*S(a[1])*S(t[2])*S(a[2])));
}

float dy3(t,a,al)
float *t,*a,*al;
{
  return(al[2]*((-S(t[0])*C(t[1])*S(t[2]))-(C(t[0])*C(a[0])*S(t[1])
         *S(t[2]))-(S(t[0])*S(t[1])*C(a[1])*C(t[2]))+(C(t[0])*C(a[0])
         *C(t[1])*C(a[1])*C(t[2]))-(C(t[0])*S(a[0])*S(a[1])*C(t[2]))));
}

float dfz2(t,a)
float *t,*a;
{
  return((S(a[0])*C(t[1])*C(t[2]))-(S(a[0])*S(t[1])*C(a[1])*S(t[2])));
}

float dgz2(t,a)
float *t,*a;
{
  return((-S(a[0])*C(t[1])*S(t[2])*C(a[2]))-(S(a[0])*S(t[1])*C(a[1])
         *C(t[2])*C(a[2]))+(S(a[0])*S(t[1])*S(a[1])*S(a[2])));
}

float dhz2(t,a)
float *t,*a;
{
  return((S(a[0])*C(t[1])*S(t[2])*S(a[2]))+(S(a[0])*S(t[1])*C(a[1])
         *C(t[2])*S(a[2]))+(S(a[0])*S(t[1])*S(a[1])*C(a[2])));
}

float dz2(t,a,d,al)
float *t,*a,*al,*d;
{
  return((al[2]*((S(a[0])*C(t[1])*C(t[2]))-(S(a[0])*S(t[1])*C(a[1])
         *S(t[2]))))+(d[2]*S(a[0])*S(t[1])*S(a[1]))+(al[1]*S(a[0])*C(t[1])));
}

float dfz3(t,a)
float *t,*a;
{
  return((-S(a[0])*S(t[1])*S(t[2]))+(S(a[0])*C(t[1])*C(a[1])*C(t[2]))
         +(C(a[0])*S(a[1])*C(t[2])));
}

float dgz3(t,a)
float *t,*a;
{
  return((-S(a[0])*S(t[1])*C(t[2])*C(a[2]))-(S(a[0])*C(t[1])*C(a[1])
         *S(t[2])*C(a[2]))-(C(a[0])*S(a[1])*S(t[2])*C(a[2])));
}
```

```
float dhz3(t,a)
float *t,*a;
{
  return((S(a[0])*S(t[1])*C(t[2])*S(a[2]))+(S(a[0])*C(t[1])*C(a[1])
        *S(t[2])*S(a[2]))+(C(a[0])*S(a[1])*S(t[2])*S(a[2])));
}

float dz3(t,a,al)
float *t,*a,*al;
{
  return(al[2]*((-S(a[0])*S(t[1])*S(t[2]))+(S(a[0])*C(t[1])*C(a[1])
        *C(t[2]))+(C(a[0])*S(a[1])*C(t[2]))));
}

float dfp4(t,a,d,al)
float *t,*a,*al,*d;
{
  return((d[5]*((-S(t[3])*S(t[4])*S(a[4]))+(C(t[3])*C(a[3])*C(t[4])
        *S(a[4]))+(C(t[3])*S(a[3])*C(a[4]))))÷(al[4]*((-S(t[3])
        *C(t[4]))-(C(t[3])*S(a[3])*S(t[4]))))+(d[4]*C(t[3])
        *S(a[3]))-(al[3]*S(t[3])));
}

float dgp4(t,a,d,al)
float *t,*a,*al,*d;
{
  return((d[5]*((C(t[3])*S(t[4])*S(a[4]))+(S(t[3])*C(a[3])*C(t[4])
        *S(a[4]))+(S(t[3])*S(a[3])*C(a[4]))))+(al[4]*((C(t[3])
        *C(t[4]))-(S(t[3])*C(a[3])*S(t[4]))))+(d[4]*S(t[3])
        *S(a[3]))+(al[3]*C(t[3])));
}

float dfp5(t,a,d,al)
float *t,*a,*al,*d;
{
  return((d[5]*((C(t[3])*C(t[4])*S(a[4]))-(S(t[3])*C(a[3])*S(t[4])
        *S(a[4]))))+(al[4]*((-C(t[3])*S(t[4]))-(S(t[3])*C(a[3])*C(t[4])))));
}

float dgp5(t,a,d,al)
float *t,*a,*al,*d;
{
  return((d[5]*((S(t[3])*C(t[4])*S(a[4]))+(C(t[3])*C(a[3])*S(t[4])
        *S(a[4]))))+(al[4]*((-S(t[3])*S(t[4]))+(C(t[3])*C(a[3])*C(t[4])))));
}

float dhp5(t,a,d,al)
float *t,*a,*al,*d;
{
  return((d[5]*S(a[3])*S(t[4])*S(a[4]))+(al[4]*S(a[3])*C(t[4])));
}

float ft6(t,a)
float *t,*a;
{
  return((C(t[3])*S(t[4])*S(a[4]))+(S(t[3])*C(a[3])*C(t[4])*S(a[4]))
        +(S(t[3])*S(a[3])*C(a[4])));
}
```

```
float gt6(t,a)
float *t,*a;
{
   return((S(t[3])*S(t[4])*S(a[4]))-(C(t[3])*C(a[3])*C(t[4])*S(a[4]))
          -(C(t[3])*S(a[3])*C(a[4])));
}

float ht6(t,a)
float *t,*a;
{
  return((-S(a[3])*C(t[4])*S(a[4]))+(C(a[3])*C(a[4])));
}
```

```
/*  ================================================================
    *            SOUS-PROGRAMME DE CALCUL LA MATRICE JACOBIENNE      *
    *                      POUR TROIS ARTICULATIONS                  *
    ================================================================ */

#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)

extern float fp(float *t,float *a,float *d,float *al);
extern float gp(float *t,float *a,float *d,float *al);
extern float hp(float *t,float *a,float *d,float *al);
extern float dfx1(float *t,float *a);
extern float dgx1(float *t,float *a);
extern float dhx1(float *t,float *a);
extern float dx1(float *t,float *a,float *d,float *al);
extern float dfx2(float *t,float *a);
extern float dgx2(float *t,float *a);
extern float dhx2(float *t,float*a);
extern float dx2(float *t,float *a,float *d,float *al);
extern float dfx3(float *t,float *a);
extern float dgx3(float *t,float *a);
extern float dhx3(float *t,float *a);
extern float dx3(float *t,float *a,float *al);
extern float dfy1(float *t,float *a);
extern float dgy1(float *t,float *a);
extern float dhy1(float *t,float *a);
extern float dy1(float *t,float *a,float *d,float *al);
extern float dfy2(float *t,float *a);
extern float dgy2(float *t,float *a);
extern float dhy2(float *t,float *a);
extern float dy2(float *t,float *a,float *d,float *al);
extern float dfy3(float *t,float *a);
extern float dgy3(float *t,float *a);
extern float dhy3(float *t,float *a);
extern float dy3(float *t,float *a,float *al);
extern float dfz2(float *t,float *a);
extern float dgz2(float *t,float *a);
extern float dhz2(float *t,float *a);
extern float dz2(float *t,float *a,float *d,float *al);
extern float dfz3(float *t,float *a);
extern float dgz3(float *t,float *a);
extern float dhz3(float *t,float *a);
extern float dz3(float *t,float *a,float *al);
extern float dfp4(float *t,float *a,float *d,float *al);
extern float dgp4(float *t,float *a,float *d,float *al);
extern float dfp5(float *t,float *a,float *d,float *al);
extern float dgp5(float *t,float *a,float *d,float *al);
extern float dhp5(float *t,float *a,float *d,float *al);
```

```
extern float fx(float *t,float *a);
extern float gx(float *t,float *a);
extern float hx(float *t,float *a);
extern float fy(float *t,float *a);
extern float gy(float *t,float *a);
extern float hy(float *t,float *a);
extern float fz(float *t,float *a);
extern float gz(float *t,float *a);
extern float hz(float *t,float *a);
extern float ft6(float *t,float *a);
extern float gt6(float *t,float *a);
extern float ht6(float *t,float *a);

extern int rep,m;
extern int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
extern float *Tab,**Q,*t,*a,*al,*d,*Tt,*Ttmax,STEP,*pas_bal;
extern float *J[3];
extern int *Ct,*Cr;

jacobi3()
{

FILE *resul3;
float Dx,Dy,Dz,R,Rmax,r,maxR;


                R=1;
                maxR=0;
                    for (Tt[2]=0; Tt[2]<=Ttmax[2]; Tt[2]+=pas_bal[2])

                     {
                       for (Tt[1]=0; Tt[1]<=Ttmax[1]; Tt[1]+=pas_bal[1])

                        {
                         for (Tt[0]=0; Tt[0]<=Ttmax[0]; Tt[0]+=pas_bal[0])

                           {
                             for (i=0; i<n; i++)
                              {
                              if (rep==0)
                              goto rotation;
                              else
                              for (j=0; j<Nb_trans; j++)
                                {
                                if (Trans[j]!=i) continue;
                                d[i]=Tt[i];
                                Ct[i]=1;
                                Cr[i]=0;
                                break;
                                }
                              if (Ct[i]==1) continue;
                    rotation: t[i]=Tt[i];
                              Ct[i]=0;
                              Cr[i]=1;

                       }
```

```
            J[0][0]=Cr[0]*dx1(t,a,d,al);

            J[0][1]=(Cr[1]*dx2(t,a,d,al))+(Ct[1]*S(t[0])*S(a[0]));

            J[0][2]=(Cr[2]*dx3(t,a,al))+(Ct[2]*((C(t[0])*S(t[1])*S(a[1]))
                    +(S(t[0])*C(a[0])*C(t[1])*S(a[1]))+(S(t[0])*S(a[0])
                    *C(a[1]))));

            J[1][0]=Cr[0]*dy1(t,a,d,al);

            J[1][1]=(Cr[1]*dy2(t,a,d,al))-(Ct[1]*C(t[0])*S(a[0]));

            J[1][2]=(Cr[2]*dy3(t,a,al))+(Ct[2]*((S(t[0])*S(t[1])*S(a[1]))
                   -(C(t[0])*C(a[0])*C(t[1])*S(a[1]))-(C(t[0])*S(a[0])
                   *C(a[1]))));

            J[2][0]=Ct[0];

            J[2][1]=(Cr[1]*dz2(t,a,d,al))+(Ct[1]*C(a[0]));

            J[2][2]=(Cr[2]*dz3(t,a,al))+(Ct[2]*((-S(a[0])*C(t[1])*S(a[1]))
                     +(C(a[0])*C(a[1]))));

            Rmax=0;

          for (i=0; i<(pow(3,n)); i++)
          {
          for (j=0; j<n; j++)
             {
            if ((t[j]+Q[i][j])>Ttmax[j])
             {
             R=0;
             goto test;
             }
             }


          Dx=((J[0][0]*Q[i][0])+(J[0][1]*Q[i][1])+(J[0][2]*Q[i][2]));

          Dy=((J[1][0]*Q[i][0])+(J[1][1]*Q[i][1])+(J[1][2]*Q[i][2]));

          Dz=((J[2][0]*Q[i][0])+(J[2][1]*Q[i][1])+(J[2][2]*Q[i][2]));


          R=sqrt((pow(Dx,2))+(pow(Dy,2))+(pow(Dz,2)));

test:     if (R>Rmax)
          Rmax=R;
          }
          if (Rmax>maxR)
          maxR=Rmax;
              }
             }
           }
          resul3=fopen("resul3","w+");
          fprintf(resul3,"maxR = %g",maxR);
          printf("\n Deplacement elementaire maximum= %g",maxR);
          fclose(resul3);
         }
```

```
/*  ======================================================================
    *            SOUS-PROGRAMME DE CALCUL LA MATRICE JACOBIENNE          *
    *                     POUR QUATRE ARTICULATIONS                      *
    ====================================================================== */

#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)

extern float fp(float *t,float *a,float *d,float *al);
extern float gp(float *t,float *a,float *d,float *al);
extern float hp(float *t,float *a,float *d,float *al);
extern float dfx1(float *t,float *a);
extern float dgx1(float *t,float *a);
extern float dhx1(float *t,float *a);
extern float dx1(float *t,float *a,float *d,float *al);
extern float dfx2(float *t,float *a);
extern float dgx2(float *t,float *a);
extern float dhx2(float *t,float*a);
extern float dx2(float *t,float *a,float *d,float *al);
extern float dfx3(float *t,float *a);
extern float dgx3(float *t,float *a);
extern float dhx3(float *t,float *a);
extern float dx3(float *t,float *a,float *al);
extern float dfy1(float *t,float *a);
extern float dgy1(float *t,float *a);
extern float dhy1(float *t,float *a);
extern float dy1(float *t,float *a,float *d,float *al);
extern float dfy2(float *t,float *a);
extern float dgy2(float *t,float *a);
extern float dhy2(float *t,float *a);
extern float dy2(float *t,float *a,float *d,float *al);
extern float dfy3(float *t,float *a);
extern float dgy3(float *t,float *a);
extern float dhy3(float *t,float *a);
extern float dy3(float *t,float *a,float *al);
extern float dfz2(float *t,float *a);
extern float dgz2(float *t,float *a);
extern float dhz2(float *t,float *a);
extern float dz2(float *t,float *a,float *d,float *al);
extern float dfz3(float *t,float *a);
extern float dgz3(float *t,float *a);
extern float dhz3(float *t,float *a);
extern float dz3(float *t,float *a,float *al);
extern float dfp4(float *t,float *a,float *d,float *al);
extern float dgp4(float *t,float *a,float *d,float *al);
extern float dfp5(float *t,float *a,float *d,float *al);
extern float dgp5(float *t,float *a,float *d,float *al);
extern float dhp5(float *t,float *a,float *d,float *al);
extern float fx(float *t,float *a);
```

```
extern float gx(float *t,float *a);
extern float hx(float *t,float *a);
extern float fy(float *t,float *a);
extern float gy(float *t,float *a);
extern float hy(float *t,float *a);
extern float fz(float *t,float *a);
extern float gz(float *t,float *a);
extern float hz(float *t,float *a);
extern float ft6(float *t,float *a);
extern float gt6(float *t,float *a);
extern float ht6(float *t,float *a);

extern int rep,m;
extern int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
extern float *Tab,**Q,*t,*a,*al,*d,*Tt,*Ttmax,STEP,*pas_bal;
extern float *J[3];
extern int *Ct,*Cr;

jacobi4()
{
FILE *resul4;

float Dx4,Dy4,Dz4,R4,Rmax4,r,maxR;


          R4=1;
          maxR=0;
          for (Tt[3]=0; Tt[3]<=Ttmax[3]; Tt[3]+=pas_bal[3])

                  {
                    for (Tt[2]=0; Tt[2]<=Ttmax[2]; Tt[2]+=pas_bal[2])

                     {
                       for (Tt[1]=0; Tt[1]<=Ttmax[1]; Tt[1]+=pas_bal[1])

                        {
                         for (Tt[0]=0; Tt[0]<=Ttmax[0]; Tt[0]+=pas_bal[0])

                           {
                             for (i=0; i<n; i++)
                              {
                              if (rep==0)
                              goto rotation;
                              else
                              for (j=0; j<Nb_trans; j++)
                                {
                                if (Trans[j]!=i) continue;
                                d[i]=Tt[i];
                                Ct[i]=1;
                                Cr[i]=0;
                                break;
                                }
                              if (Ct[i]==1) continue;
                    rotation: t[i]=Tt[i];
                              Ct[i]=0;
                              Cr[i]=1;

                      }
```

```
J[0][0]=Cr[0]*((dfx1(t,a)*fp(t,a,d,al))+(dgx1(t,a)*gp(t,a,d,al))
        +(dhx1(t,a)*hp(t,a,d,al))+dx1(t,a,d,al));

J[0][1]=Cr[1]*((dfx2(t,a)*fp(t,a,d,al))+(dgx2(t,a)*gp(t,a,d,al))
        +(dhx2(t,a)*hp(t,a,d,al))+dx2(t,a,d,al))+(Ct[1]*S(t[0])*S(a[0]));

J[0][2]=Cr[2]*((dfx3(t,a)*fp(t,a,d,al))+(dgx3(t,a)*gp(t,a,d,al))
        +(dhx3(t,a)*hp(t,a,d,al))+dx3(t,a,al))+(Ct[2]*((C(t[0])
        *S(t[1])*S(a[1]))+(S(t[0])*C(a[0])*C(t[1])*S(a[1]))+(S(t[0])
        *S(a[0])*C(a[1]))));

J[0][3]=Cr[3]*((fx(t,a)*dfp4(t,a,d,al))+(gx(t,a)*dgp4(t,a,d,al)))
        +(Ct[3]*hx(t,a));

J[1][0]=Cr[0]*((dfy1(t,a)*fp(t,a,d,al))+(dgy1(t,a)*gp(t,a,d,al))
        +(dhy1(t,a)*hp(t,a,d,al))+dy1(t,a,d,al));

J[1][1]=Cr[1]*((dfy2(t,a)*fp(t,a,d,al))+(dgy2(t,a)*gp(t,a,d,al))
        +(dhy2(t,a)*hp(t,a,d,al))+dy2(t,a,d,al))+(Ct[1]*(-C(t[0])*S(a[0])));

J[1][2]=Cr[2]*((dfy3(t,a)*fp(t,a,d,al))+(dgy3(t,a)*gp(t,a,d,al))
        +(dhy3(t,a)*hp(t,a,d,al))+dy3(t,a,al))+(Ct[2]*((S(t[0])
        *S(t[1])*S(a[1]))-(C(t[0])*C(a[0])*C(t[1])*S(a[1]))-(C(t[0])
        *S(a[0])*C(a[1]))));

J[1][3]=Cr[3]*((fy(t,a)*dfp4(t,a,d,al))+(gy(t,a)*dgp4(t,a,d,al)))
        +(Ct[3]*hy(t,a));

J[2][0]=Ct[0];

J[2][1]=Cr[1]*((dfz2(t,a)*fp(t,a,d,al))+(dgz2(t,a)*gp(t,a,d,al))
        +(dhz2(t,a)*hp(t,a,d,al))+dz2(t,a,d,al))+(Ct[1]*C(a[0]));

J[2][2]=Cr[2]*((dfz3(t,a)*fp(t,a,d,al))+(dgz3(t,a)*gp(t,a,d,al))
        +(dhz3(t,a)*hp(t,a,d,al))+dz3(t,a,al))+(Ct[2]*((-S(a[0])
        *C(t[1])*S(a[1]))+(C(a[0])*C(a[1]))));

J[2][3]=Cr[3]*((fz(t,a)*dfp4(t, a,d,al))+(gz(t,a)*dgp4(t,a,d,al)))
        +(Ct[3]*hz(t,a));

        Rmax4=0;

        for (i=0; i<(pow(3,n)); i++)
        {
        for (j=0; j<n; j++)
          {
           if ((t[j]+Q[i][j])>Ttmax[j])
           {
           R4=0;
           goto test2;
           }
           }

        Dx4=((J[0][0]*Q[i][0])+(J[0][1]*Q[i][1])+(J[0][2]*Q[i][2])
            +(J[0][3]*Q[i][3]));

        Dy4=((J[1][0]*Q[i][0])+(J[1][1]*Q[i][1])+(J[1][2]*Q[i][2])
            +(J[1][3]*Q[i][3]));
```

```
        Dz4=((J[2][0]*Q[i][0])+(J[2][1]*Q[i][1])+(J[2][2]*Q[i][2])
            +(J[2][3]*Q[i][3]));


        R4=sqrt((pow(Dx4,2))+(pow(Dy4,2))+(pow(Dz4,2)));

test2:  if (R4>Rmax4)
        Rmax4=R4;

        }
        if (Rmax4>maxR)
        maxR=Rmax4;
            }
           }
         }
       }
       resul4=fopen("resul4","w+");
       fprintf(resul4,"maxR= %g",maxR);
       printf("\n Deplacement elementaire maximum = %g",maxR);
       fclose(resul4);
    }
```

```
/*  ========================================================================
    *            SOUS-PROGRAMME DE CALCUL LA MATRICE JACOBIENNE            *
    *                        POUR CINQ ARTICULATIONS                       *
    ======================================================================== */

#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)

extern float fp(float *t,float *a,float *d,float *al);
extern float gp(float *t,float *a,float *d,float *al);
extern float hp(float *t,float *a,float *d,float *al);
extern float dfx1(float *t,float *a);
extern float dgx1(float *t,float *a);
extern float dhx1(float *t,float *a);
extern float dx1(float *t,float *a,float *d,float *al);
extern float dfx2(float *t,float *a);
extern float dgx2(float *t,float *a);
extern float dhx2(float *t,float*a);
extern float dx2(float *t,float *a,float *d,float *al);
extern float dfx3(float *t,float *a);
extern float dgx3(float *t,float *a);
extern float dhx3(float *t,float *a);
extern float dx3(float *t,float *a,float *al);
extern float dfy1(float *t,float *a);
extern float dgy1(float *t,float *a);
extern float dhy1(float *t,float *a);
extern float dy1(float *t,float *a,float *d,float *al);
extern float dfy2(float *t,float *a);
extern float dgy2(float *t,float *a);
extern float dhy2(float *t,float *a);
extern float dy2(float *t,float *a,float *d,float *al);
extern float dfy3(float *t,float *a);
extern float dgy3(float *t,float *a);
extern float dhy3(float *t,float *a);
extern float dy3(float *t,float *a,float *al);
extern float dfz2(float *t,float *a);
extern float dgz2(float *t,float *a);
extern float dhz2(float *t,float *a);
extern float dz2(float *t,float *a,float *d,float *al);
extern float dfz3(float *t,float *a);
extern float dgz3(float *t,float *a);
extern float dhz3(float *t,float *a);
extern float dz3(float *t,float *a,float *al);
extern float dfp4(float *t,float *a,float *d,float *al);
extern float dgp4(float *t,float *a,float *d,float *al);
extern float dfp5(float *t,float *a,float *d,float *al);
extern float dgp5(float *t,float *a,float *d,float *al);
extern float dhp5(float *t,float *a,float *d,float *al);
extern float fx(float *t,float *a);
extern float gx(float *t,float *a);
```

```
extern float hx(float *t,float *a);
extern float fy(float *t,float *a);
extern float gy(float *t,float *a);
extern float hy(float *t,float *a);
extern float fz(float *t,float *a);
extern float gz(float *t,float *a);
extern float hz(float *t,float *a);
extern float ft6(float *t,float *a);
extern float gt6(float *t,float *a);
extern float ht6(float *t,float *a);

extern int rep,m;
extern int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
extern float *Tab,**Q,*t,*a,*al,*d,*Tt,*Ttmax,STEP,*pas_bal;
extern float *J[3];
extern int *Ct,*Cr;

jacobi5()
{
FILE *resul5;
float Dx5,Dy5,Dz5,R5,Rmax5,r,maxR;


          R5=1;
          maxR=0;
               for (Tt[4]=0; Tt[4]<=Ttmax[4]; Tt[4]+=pas_bal[4])

                {
                  for (Tt[3]=0; Tt[3]<=Ttmax[3]; Tt[3]+=pas_bal[3])

                   {
                     for (Tt[2]=0; Tt[2]<=Ttmax[2]; Tt[2]+=pas_bal[2])

                      {
                        for (Tt[1]=0; Tt[1]<=Ttmax[1]; Tt[1]+=pas_bal[1])

                         {
                          for (Tt[0]=0; Tt[0]<=Ttmax[0]; Tt[0]+=pas_bal[0])

                            {
                              for (i=0; i<n; i++)
                               {
                               if (rep==0)
                               goto rotation;
                               else
                               for (j=0; j<Nb_trans; j++)
                                 {
                                 if (Trans[j]!=i) continue;
                                 d[i]=Tt[i];
                                 Ct[i]=1;
                                 Cr[i]=0;
                                 break;
                                 }
                               if (Ct[i]==1) continue;
                          rotation: t[i]=Tt[i];
                                    Ct[i]=0;
                                    Cr[i]=1;
                            }
```

```
J[0][0]=Cr[0]*((dfx1(t,a)*fp(t,a,d,al))+(dgx1(t,a)*gp(t,a,d,al))
        +(dhx1(t,a)*hp(t,a,d,al))+dx1(t,a,d,al));

J[0][1]=Cr[1]*((dfx2(t,a)*fp(t,a,d,al))+(dgx2(t,a)*gp(t,a,d,al))
        +(dhx2(t,a)*hp(t,a,d,al))+dx2(t,a,d,al))+(Ct[1]*S(t[0])*S(a[0]));

J[0][2]=Cr[2]*((dfx3(t,a)*fp(t,a,d,al))+(dgx3(t,a)*gp(t,a,d,al))
        +(dhx3(t,a)*hp(t,a,d,al))+dx3(t,a,al))+(Ct[2]*((C(t[0])
        *S(t[1])*S(a[1]))+(S(t[0])*C(a[0])*C(t[1])*S(a[1]))
        +(S(t[0])*S(a[0])*C(a[1]))));

J[0][3]=Cr[3]*((fx(t,a)*dfp4(t,a,d,al))+(gx(t,a)*dgp4(t,a,d,al)))
        +(Ct[3]*hx(t,a));

J[0][4]=Cr[4]*((fx(t,a)*dfp5(t,a,d,al))+(gx(t,a)*dgp5(t,a,d,al))
        +(hx(t,a)*dhp5(t,a,d,al)))+(Ct[4]*((fx(t,a)*S(t[3])*S(a[3]))
        -(gx(t,a)*C(t[3])*S(a[3]))-(hx(t,a)*C(a[3]))));

J[1][0]=Cr[0]*((dfy1(t,a)*fp(t,a,d,al))+(dgy1(t,a)*gp(t,a,d,al))
        +(dhy1(t,a)*hp(t,a,d,al))+dy1(t,a,d,al));

J[1][1]=Cr[1]*((dfy2(t,a)*fp(t,a,d,al))+(dgy2(t,a)*gp(t,a,d,al))
        +(dhy2(t,a)*hp(t,a,d,al))+dy2(t,a,d,al))+(Ct[1]*(-C(t[0])*S(a[0])));

J[1][2]=Cr[2]*((dfy3(t,a)*fp(t,a,d,al))+(dgy3(t,a)*gp(t,a,d,al))
        +(dhy3(t,a)*hp(t,a,d,al))+dy3(t,a,al))+(Ct[2]*((S(t[0])
        *S(t[1])*S(a[1]))-(C(t[0])*C(a[0])*C(t[1])*S(a[1]))
        -(C(t[0])*S(a[0])*C(a[1]))));

J[1][3]=Cr[3]*((fy(t,a)*dfp4(t,a,d,al))+(gy(t,a)*dgp4(t,a,d,al)))
        +(Ct[3]*hy(t,a));

J[1][4]=Cr[4]*((fy(t,a)*dfp5(t,a,d,al))+(gy(t,a)*dgp5(t,a,d,al))
        +(hy(t,a)*dhp5(t,a,d,al)))+(Ct[4]*((fy(t,a)*S(t[3])*S(a[3]))
        -(gy(t,a)*C(t[3])*S(a[3]))-(hy(t,a)*C(a[3]))));

J[2][0]=Ct[0];

J[2][1]=Cr[1]*((dfz2(t,a)*fp(t,a,d,al))+(dgz2(t,a)*gp(t,a,d,al))
        +(dhz2(t,a)*hp(t,a,d,al))+dz2(t,a,d,al))+(Ct[1]*C(a[0]));

J[2][2]=Cr[2]*((dfz3(t,a)*fp(t,a,d,al))+(dgz3(t,a)*gp(t,a,d,al))
        +(dhz3(t,a)*hp(t,a,d,al))+dz3(t,a,al))+(Ct[2]*((-S(a[0])
        *C(t[1])*S(a[1]))+(C(a[0])*C(a[1]))));

J[2][3]=Cr[3]*((fz(t,a)*dfp4(t, a,d,al))+(gz(t,a)*dgp4(t,a,d,al)))
        +(Ct[3]*hz(t,a));

J[2][4]=Cr[4]*((fz(t,a)*dfp5(t,a,d,al))+(gz(t,a)*dgp5(t,a,d,al))
        +(hz(t,a)*dhp5(t,a,d,al)))+(Ct[4]*((fz(t,a)*S(t[3])*S(a[3]))
        -(gz(t,a)*C(t[3])*S(a[3]))-(hz(t,a)*C(a[3]))));


        Rmax5=0;
```

```
            for (i=0; i<(pow(3,n)); i++)
            {
            for (j=0; j<n; j++)
               {
               if ((t[j]+Q[i][j])>Ttmax[j])
               {
               R5=0;
               goto test3;
               }
               }

            Dx5=((J[0][0]*Q[i][0])+(J[0][1]*Q[i][1])+(J[0][2]*Q[i][2])
                 +(J[0][3]*Q[i][3])+(J[0][4]*Q[i][4]));

            Dy5=((J[1][0]*Q[i][0])+(J[1][1]*Q[i][1])-(J[1][2]*Q[i][2])
                 +(J[1][3]*Q[i][3])+(J[1][4]*Q[i][4]));

            Dz5=((J[2][0]*Q[i][0])+(J[2][1]*Q[i][1])+(J[2][2]*Q[i][2])
                 +(J[2][3]*Q[i][3])+(J[2][4]*Q[i][4]));

            R5=sqrt((pow(Dx5,2))+(pow(Dy5,2))+(pow(Dz5,2)));

test3:      if (R5>Rmax5)
            Rmax5=R5;

            }
            if (Rmax5>maxR)
            maxR=Rmax5;
            printf("\n %g\n",maxR);
                }
               }
              }
            }
            resul5=fopen("resul5","w+");
            fprintf(resul5,"maxR= %g",maxR);
            printf("\n Deplacement elementaire maximum = %g",maxR);
            fclose(resul5);
        }
```

```
/*  ================================================================
    *            SOUS-PROGRAMME DE CALCUL LA MATRICE JACOBIENNE      *
    *                        POUR SIX ARTICULATIONS                  *
    ================================================================ */

#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)

extern float fp(float *t,float *a,float *d,float *al);
extern float gp(float *t,float *a,float *d,float *al);
extern float hp(float *t,float *a,float *d,float *al);
extern float dfx1(float *t,float *a);
extern float dgx1(float *t,float *a);
extern float dhx1(float *t,float *a);
extern float dx1(float *t,float *a,float *d,float *al);
extern float dfx2(float *t,float *a);
extern float dgx2(float *t,float *a);
extern float dhx2(float *t,float*a);
extern float dx2(float *t,float *a,float *d,float *al);
extern float dfx3(float *t,float *a);
extern float dgx3(float *t,float *a);
extern float dhx3(float *t,float *a);
extern float dx3(float *t,float *a,float *al);
extern float dfy1(float *t,float *a);
extern float dgy1(float *t,float *a);
extern float dhy1(float *t,float *a);
extern float dy1(float *t,float *a,float *d,float *al);
extern float dfy2(float *t,float *a);
extern float dgy2(float *t,float *a);
extern float dhy2(float *t,float *a);
extern float dy2(float *t,float *a,float *d,float *al);
extern float dfy3(float *t,float *a);
extern float dgy3(float *t,float *a);
extern float dhy3(float *t,float *a);
extern float dy3(float *t,float *a,float *al);
extern float dfz2(float *t,float *a);
extern float dgz2(float *t,float *a);
extern float dhz2(float *t,float *a);
extern float dz2(float *t,float *a,float *d,float *al);
extern float dfz3(float *t,float *a);
extern float dgz3(float *t,float *a);
extern float dhz3(float *t,float *a);
extern float dz3(float *t,float *a,float *al);
extern float dfp4(float *t,float *a,float *d,float *al);
extern float dgp4(float *t,float *a,float *d,float *al);
extern float dfp5(float *t,float *a,float *d,float *al);
extern float dgp5(float *t,float *a,float *d,float *al);
extern float dhp5(float *t,float *a,float *d,float *al);
extern float fx(float *t,float *a);
extern float gx(float *t,float *a);
```

```
extern float hx(float *t,float *a);
extern float fy(float *t,float *a);
extern float gy(float *t,float *a);
extern float hy(float *t,float *a);
extern float fz(float *t,float *a);
extern float gz(float *t,float *a);
extern float hz(float *t,float *a);
extern float ft6(float *t,float *a);
extern float gt6(float *t,float *a);
extern float ht6(float *t,float *a);

extern int rep,m;
extern int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
extern float *Tab,**Q,*t,*a,*al,*d,*Tt,*Ttmax,STEP,*pas_bal;
extern float *J[3];
extern int *Ct,*Cr;

jacobi6()
{
float Dx6,Dy6,Dz6,R6,Rmax6,r,maxR;
FILE *resul6;

          R6=1;
          maxR=0;
          for (Tt[5]=0; Tt[5]<=Ttmax[5]; Tt[5]+=pas_bal[5])

             {
               for (Tt[4]=0; Tt[4]<=Ttmax[4]; Tt[4]+=pas_bal[4])

                {
                  for (Tt[3]=0; Tt[3]<=Ttmax[3]; Tt[3]+=pas_bal[3])

                   {
                     for (Tt[2]=0; Tt[2]<=Ttmax[2]; Tt[2]+=pas_bal[2])

                      {
                        for (Tt[1]=0; Tt[1]<=Ttmax[1]; Tt[1]+=pas_bal[1])

                         {
                          for (Tt[0]=0; Tt[0]<=Ttmax[0]; Tt[0]+=pas_bal[0])

                            {
                              for (i=0; i<n; i++)
                               {
                               if (rep==0)
                               goto rotation;
                               else
                               for (j=0; j<Nb_trans; j++)
                                 {
                                 if (Trans[j]!=i) continue;
                                 d[i]=Tt[i];
                                 Ct[i]=1;
                                 Cr[i]=0;
                                 break;
                                 }
                               if (Ct[i]==1) continue;
                     rotation: t[i]=Tt[i];
                               Ct[i]=0;
                               Cr[i]=1;
                               }
```

```
J[0][0]=Cr[0]*((dfx1(t,a)*fp(t,a,d,al))+(dgx1(t,a)*gp(t,a,d,al))
       +(dhx1(t,a)*hp(t,a,d,al))+dx1(t,a,d,al));

J[0][1]=Cr[1]*((dfx2(t,a)*fp(t,a,d,al))+(dgx2(t,a)*gp(t,a,d,al))
       +(dhx2(t,a)*hp(t,a,d,al))+dx2(t,a,d,al))+(Ct[1]*S(t[0])*S(a[0]));

J[0][2]=Cr[2]*((dfx3(t,a)*fp(t,a,d,al))+(dgx3(t,a)*gp(t,a,d,al))
       +(dhx3(t,a)*hp(t,a,d,al))+dx3(t,a,al))+(Ct[2]*((C(t[0])
       *S(t[1])*S(a[1]))+(S(t[0])*C(a[0])*C(t[1])*S(a[1]))
       +(S(t[0])*S(a[0])*C(a[1]))));

J[0][3]=Cr[3]*((fx(t,a)*dfp4(t,a,d,al))+(gx(t,a)*dgp4(t,a,d,al)))
       +(Ct[3]*hx(t,a));

J[0][4]=Cr[4]*((fx(t,a)*dfp5(t,a,d,al))+(gx(t,a)*dgp5(t,a,d,al))
       +(hx(t,a)*dhp5(t,a,d,al)))+(Ct[4]*((fx(t,a)*S(t[3])*S(a[3]))
       -(gx(t,a)*C(t[3])*S(a[3]))-(hx(t,a)*C(a[3]))));

J[0][5]=Ct[5]*((fx(t,a)*ft6(t,a))+(gx(t,a)*gt6(t,a))+(hx(t,a)*ht6(t,a)));

J[1][0]=Cr[0]*((dfy1(t,a)*fp(t,a,d,al))+(dgy1(t,a)*gp(t,a,d,al))
       +(dhy1(t,a)*hp(t,a,d,al))+dy1(t,a,d,al));

J[1][1]=Cr[1]*((dfy2(t,a)*fp(t,a,d,al))+(dgy2(t,a)*gp(t,a,d,al))
       +(dhy2(t,a)*hp(t,a,d,al))+dy2(t,a,d,al))+(Ct[1]*(-C(t[0])*S(a[0])));

J[1][2]=Cr[2]*((dfy3(t,a)*fp(t,a,d,al))+(dgy3(t,a)*gp(t,a,d,al))
       +(dhy3(t,a)*hp(t,a,d,al))+dy3(t,a,al))+(Ct[2]*((S(t[0])
       *S(t[1])*S(a[1]))-(C(t[0])*C(a[0])*C(t[1])*S(a[1]))
       -(C(t[0])*S(a[0])*C(a[1]))));

J[1][3]=Cr[3]*((fy(t,a)*dfp4(t,a,d,al))+(gy(t,a)*dgp4(t,a,d,al)))
       +(Ct[3]*hy(t,a));

J[1][4]=Cr[4]*((fy(t,a)*dfp5(t,a,d,al))+(gy(t,a)*dgp5(t,a,d,al))
       +(hy(t,a)*dhp5(t,a,d,al)))+(Ct[4]*((fy(t,a)*S(t[3])*S(a[3]))
       -(gy(t,a)*C(t[3])*S(a[3]))-(hy(t,a)*C(a[3]))));

J[1][5]=Ct[5]*((fy(t,a)*ft6(t,a))+(gy(t,a)*gt6(t,a))+(hy(t,a)*ht6(t,a)));

J[2][0]=Ct[0];

J[2][1]=Cr[1]*((dfz2(t,a)*fp(t,a,d,al))+(dgz2(t,a)*gp(t,a,d,al))
       +(dhz2(t,a)*hp(t,a,d,al))+dz2(t,a,d,al))+(Ct[1]*C(a[0]));

J[2][2]=Cr[2]*((dfz3(t,a)*fp(t,a,d,al))+(dgz3(t,a)*gp(t,a,d,al))
       +(dhz3(t,a)*hp(t,a,d,al))+dz3(t,a,al))+(Ct[2]*((-S(a[0])
       *C(t[1])*S(a[1]))+(C(a[0])*C(a[1]))));

J[2][3]=Cr[3]*((fz(t,a)*dfp4(t, a,d,al))+(gz(t,a)*dgp4(t,a,d,al)))
       +(Ct[3]*hz(t,a));

J[2][4]=Cr[4]*((fz(t,a)*dfp5(t,a,d,al))+(gz(t,a)*dgp5(t,a,d,al))
       +(hz(t,a)*dhp5(t,a,d,al)))+(Ct[4]*((fz(t,a)*S(t[3])*S(a[3]))
       -(gz(t,a)*C(t[3])*S(a[3]))-(hz(t,a)*C(a[3]))));

J[2][5]=Ct[5]*((fz(t,a)*ft6(t,a))+(gz(t,a)*gt6(t,a))+(hz(t,a)*ht6(t,a)));
```

```
            Rmax6=0;

            for (i=0; i<(pow(3,n)); i++)
            {
            for (j=0; j<n; j++)
               {
               if ((t[j]+Q[i][j])>Ttmax[j])
               {
               R6=0;
               goto test4;
               }
               }

            Dx6=((J[0][0]*Q[i][0])+(J[0][1]*Q[i][1])+(J[0][2]*Q[i][2])
                +(J[0][3]*Q[i][3])+(J[0][4]*Q[i][4])+(J[0][5]*Q[i][5]));

            Dy6=((J[1][0]*Q[i][0])+(J[1][1]*Q[i][1])+(J[1][2]*Q[i][2])
                +(J[1][3]*Q[i][3])+(J[1][4]*Q[i][4])+(J[1][5]*Q[i][5]));

            Dz6=((J[2][0]*Q[i][0])+(J[2][1]*Q[i][1])+(J[2][2]*Q[i][2])
                +(J[2][3]*Q[i][3])+(J[2][4]*Q[i][4])+(J[2][5]*Q[i][5]));


            R6=sqrt((pow(Dx6,2))+(pow(Dy6,2))+(pow(Dz6,2)));

test4:      if (R6>Rmax6)
            Rmax6=R6;

            }
            if (Rmax6>maxR)
            maxR=Rmax6;
            printf("\n %g\n",maxR);
                }
               }
              }
             }
           }
         }
         resul6=fopen("resul6","w+");
         fprintf(resul6,"maxR= %g",maxR);
         printf("deplacement elementaire maximum = %g",maxR);
         fclose(resul6);
}
```

ANNEXE B

Programme de génération de la trajectoire

```
/*    ================================================================
      *                      PROGRAMME PRINCIPAL                       *
      *                                                                *
      *      de simulation de la generation de la trajectoire          *
      *                                                                *
      *               Par Josiane JUEGOUO. 1993                        *
      ================================================================   */


#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)



int rep,m,iter;
int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
float *t,*a,*al,*d,*Tt,*Ttmax,STEP,*pas_bal;
float *J[6],Dx[729],Dy[729],Dz[729],DOx[729],DOy[729],DOz[729];
float  Epx,Epy,Epz,Eox,Eoy,Eoz;
int Ct,Cr,ind,ind1,ind2;
float **Q,*Tab,T[4][4],Erx,Ery,Erz,Erox,Eroy,Eroz,ex,ey,ez,eox,eoy,eoz;
float Mx,My,Mz,Mox,Moy,Moz,M1x,M1y,M1z,M1ox,M1oy,M1oz;
float M2x,M2y,M2z,M2ox,M2oy,M2oz;
float stepR,stepT,ob1,ob2,ob3,ob4,ob5,ob6,ar,br,cr,R0,R1,Ri;
float m1,m2,m3,m4,m5,m6,Ar,Br,Dr;
double u1,v1,w1,u2,v2,w2,ui,vi,wi;
int fact,test3;
double ECR,THETA=.25,Tet1,Teti;
float eps1=0.075;             /*Critere d'arret de positionnement */
float eps2=0.00150;           /*Critere d'arret d'orientation*/

main()
{
printf("INTRODUCTION DES PARAMETRES CONSTANTS\n");

/*************************************************************
 * Il s'agit d'introduire le nombre d'articulations du bras,  *
 * de preciser leur nature et d'entrer les parametres du bras *
 *************************************************************/

ind=0;
printf("entrer le nombre d'articulations n:  ");fflush(stdout);

scanf("%d",&n);
Nb_comb=(pow(3,n));
printf("\n   Nombre d'articulations : %d\n\n",n);
printf("   Nombre de combinaisons : %d\n\n",Nb_comb);
printf("   Y a-t-il des articulations qui sont des translations?\n");
printf("\n   Si oui taper 1, si non taper 0:  ");fflush(stdout);
scanf("%d",&rep);
```

```
switch(rep)
   {
   case 1: printf("\n  Entrer le nombre de translations :  ");fflush(stdout);
           scanf("%d",&Nb_trans);
           printf("\n  Entrer le pas lineaire: stepT  ");fflush(stdout);
           scanf("%g",&stepT);
           printf("\n  Entrer le pas angulaire: stepR  ");fflush(stdout);
           scanf("%g",&stepR);
           break;
   case 0: Nb_trans=0;
            printf("\n  Entrer le pas angulaire: stepR   ");fflush(stdout);
            scanf("%g",&stepR);
   }

Trans=Calloc(Nb_trans,int);         /*Trans est le vecteur des translations*/

if (Nb_trans !=0)

   for (i=0; i<Nb_trans; i++)
     {
     printf("\n  Entrer le numero de la translation :  ");fflush(stdout);
     scanf("%d",&Trans[i]);
     }

/*****************************************************************************
 *INTRODUCTION DES PARAMETRES DE D-H                                         *
 *Le vecteur "t" correspond aux angles theta;                                *
 *le vecteur "a" aux angles alpha (il n'est pas utilis dans les simulations);*
 *le vecteur "al" aux constantes ai et le vecteur "d" aux constantes di .    *
 *****************************************************************************/

t=Calloc(n,TRUC);
al=Calloc(n,TRUC);
d=Calloc(n,TRUC);

printf("\n  Introduire les parametres constants et initialiser les \n");
printf("  parametres variables .\n");fflush(stdout);

printf("\n        INTRODUCTION DES PARAMETRES THETA   \n");
for(i=0; i<n; i++)
   {
     printf("  theta[ %d ]=\t",i);fflush(stdout);
     scanf("%g",&t[i]);
   }
printf("\n         INTRODUCTION DES CONSTANTES ai      \n");
for(i=0; i<n; i++)
   {
     printf("  a[ %d ]=\t",i);fflush(stdout);
     scanf("%g",&al[i]);
   }
printf("\n         INTRODUCTION DES PARAMETRES di       \n");
for (i=0; i<n; i++)
  {
    printf("  d[ %d ]=\t",i);fflush(stdout);
    scanf("%g",&d[i]);
  }

printf("\n    COORDONNEES DU POINT FINAL     \n");
```

```
/*************************************************************************
 * Les coordonnees du point final sont introduites en millimetres;       *
 * les trois premieres designent la position de l'extremite du bras,     *
 * et les trois dernieres la position de l'extremite du poignet oriente  *
 *************************************************************************/

    printf(" M2x= \t");fflush(stdout);
    scanf("%g",&M2x);
    printf(" M2y= \t");fflush(stdout);
    scanf("%g",&M2y);
    printf(" M2z= \t");fflush(stdout);
    scanf("%g",&M2z);
    printf(" M2ox= \t");fflush(stdout);
    scanf("%g",&M2ox);
    printf(" M2oy= \t");fflush(stdout);
    scanf("%g",&M2oy);
    printf(" M2oz= \t");fflush(stdout);
    scanf("%g",&M2oz);



Tab=Calloc((n*Nb_comb),TRUC);
Q=Calloc(Nb_comb,TRUC *);
if (Tab==NULL || Q==NULL) exit(1);
for (i=0; i<Nb_comb; i++)
   {
   Q[i]=Tab+i*n;
   }

/*  Allocation d'espace pour la matrice jacobienne */

 for (i=0; i<6; i++)
   {
    J[i]=Calloc(n,TRUC);
   }

          iter=1;
          matq();

          u2=(M2ox-M2x)/d[5];
          v2=(M2oy-M2y)/d[5];
          w2=(M2oz-M2z)/d[5];

        /*u2,v2,w2 sont les cosinus directeurs de l'orientation visee. */


BOUCLE :  pt_courant();

          jacobpos();

   ECR=sqrt(pow(ex,2)+pow(ey,2)+pow(ez,2));

   if(ECR<eps1)   goto boucle2;

   for (i=0; i<n; i++)
     {
     t[i]+=Q[ind][i];
     }
   iter+=1;
   goto BOUCLE;
```

```
boucle2:   if(fabs(THETA)<eps2)  exit (0); /* exit(0) est remplace par goto fin
                                           si on veut obtenir les valeurs
                                           angulaires correspondant a la
                                           position finale du jalon parcouru.*/

           for (i=0; i<n; i++)
            {
            t[i]+=Q[ind][i];
            }
            iter+=1;
           pt_courant();

           jacobori();

            goto boucle2;

fin:      for (i=0; i<n; i++)
          {
          t[i]+=Q[ind][i];
          printf("\nt[%d]=%g\n",i,t[i]);
          }
}
         /*************************************************************
          * Fonction definissant l'angle beta qui permet d'effectuer *
          * l'orientation du poignet.                                 *
          *************************************************************/

double fonc(double a2,double a1,double b2,double b1,double c2,double c1)
{
  return(2*asin(0.5*(sqrt(pow((a2-a1),2)+pow((b2-b1),2)+pow((c2-c1),2)))));
}
```

```
/*  ========================================================================= *
 *                SOUS-PROGRAMME DE CALCUL DU POINT ACTUEL                   *
 *                                                                           *
 *                     Par Josiane JUEGOUO. 1993                             *
/*  ========================================================================= */

#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)


extern int rep,m,iter;
```

```
/* ====================================================================
   * SOUS-PROGRAMME PERMETTANT DE GENERER LA MATRICE DES COMBINAISONS *
   *                                                                  *
   *                    Par Josiane JUEGOUO. 1993.                    *
   ==================================================================== */


#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)


extern int rep,m;
extern int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
extern float *Tab,**Q,*t,*a,*al,*d,*Tt,*Ttmax,STEP,stepR,stepT,*pas_bal;


matq()
{
FILE * matrice;

 /* Generation de la matrice */

if(( matrice=fopen("matrice","w+"))==NULL)
  {
  printf("Ouverture du fichier impossible \n");
  exit(1);
  }

  N=n-1;

  switch(rep)
  {
case 1:   for(j=N; j>=0; j--)
      {
       for (m=0; m<Nb_trans; m++)
       {  if (j==Trans[m])
            {
            STEP=stepT;
            break;
            }
            else
             {
               if (m==Nb_trans-1)
                STEP=stepR;
             }
        }
       s=0;
       A=pow(3,j);
       for(k=1; k<=pow(3,(N-j)); k++)
       {   for(i=s; i<=(s+(pow(3,j)-1)); i++)

           {   Q[i][j]=-STEP;
```

```
                Q[i+A][j]=0;
                Q[i+(2*A)][j]=STEP;
            }

            s=k*pow(3,j+1);
        }
    }
   break;

case 0: for (j=N; j>=0; j--)
          {
          s=0;
          A=pow(3,j);
          for (k=1; k<=pow(3,(N-j)); k++)
            {  for(i=s; i<=(s+(pow(3,j)-1)); i++)
                 {  Q[i][j]=-stepR;
                    Q[i+A][j]=0;
                    Q[i+(2*A)][j]=stepR;
                 }
                 s=k*pow(3,j+1);
            }
          }
      }
    B=Nb_comb;
    for(i=0; i<B; i++)
    {

      for(j=0; j<n; j++)
      {
              fprintf(matrice,"Q%d%d=%g  ",i,j,Q[i][j]);

      }
      fprintf(matrice,"\n");

    }
     fclose(matrice);
  }
```

```
/*  ================================================================== *
 *                SOUS-PROGRAMME DE CALCUL DU POINT ACTUEL             *
 *                                                                     *
 *                    Par Josiane JUEGOUO. 1993                        *
/*  ================================================================== */

#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)


extern int rep,m,iter;
extern int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
extern float *Tab,**Q,*t,*a,*al,*d,STEP;
extern float *J[6],M1x,M1y,M1z,M1ox,M1oy,M1oz,ob1,ob2,ob3,ob4,ob5,ob6;
extern float Epx,Epy,Epz,Eox,Eoy,Eoz,Erx,Ery,Erz,Erox,Eroy,Eroz;
extern int *Ct,*Cr;
extern float T[4][4],eps1,eps2,ex,ey,ez,eox,eoy,eoz;
double mod;

/* T est la matrice de transformation globale
 dont les elements sont les T[i][j] .
 Le vecteur M1 reprsente les coordonnes du
 point courant. */

pt_courant ()

{

    T[0][0]=C(t[0])*(C(t[1]+t[2])*((C(t[3])*C(t[4])*C(t[5]))-(S(t[3])
            *S(t[5])))-(S(t[1]+t[2])*S(t[4])*C(t[5])))-(S(t[0])*((S(t[3])
            *C(t[4])*C(t[5]))+(C(t[3])*S(t[5]))));

    T[0][1]=C(t[0])*(-C(t[1]+t[2])*((C(t[3])*C(t[4])*S(t[5]))+(S(t[3])
            *C(t[5])))-(S(t[1]+t[2])*S(t[4])*S(t[5])))-(S(t[0])*((-S(t[3])
            *C(t[4])*S(t[5]))+(C(t[3])*C(t[5]))));

    T[0][2]=C(t[0])*((C(t[1]+t[2])*C(t[3])*S(t[4]))+(S(t[1]+t[2])*C(t[4])))
            -(S(t[0])*S(t[3])*S(t[4]));

    T[0][3]=C(t[0])*(d[5]*((C(t[1]+t[2])*C(t[3])*S(t[4]))+(S(t[1]+t[2])
            *C(t[4])))+(d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2]))+(al[1]
            *C(t[1])))-(S(t[0])*(d[5]*S(t[3])*S(t[4])+d[1]));

    T[1][0]=S(t[0])*(C(t[1]+t[2])*((C(t[3])*C(t[4])*C(t[5]))-(S(t[3])*
            S(t[5])))-(S(t[1]+t[2])*S(t[4])*C(t[5])))+(C(t[0])*((S(t[3])
            *C(t[4])*C(t[5]))+(C(t[3])*S(t[5]))));
```

```
    T[1][1]=S(t[0])*(-C(t[1]+t[2])*((C(t[3])*C(t[4])*S(t[5]))+(S(t[3])
            *C(t[5])))-(S(t[1]+t[2])*S(t[4])*S(t[5])))+(C(t[0])*((-S(t[3])
            *C(t[4])*S(t[5]))+(C(t[3])*C(t[5]))));

    T[1][2]=S(t[0])*((C(t[1]+t[2])*C(t[3])*S(t[4]))+(S(t[1]+t[2])*C(t[4])))
            +(C(t[0])*S(t[3])*S(t[4]));

    T[1][3]=S(t[0])*(d[5]*((C(t[1]+t[2])*C(t[3])*S(t[4]))+(S(t[1]+t[2])
            *C(t[4])))+(d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2]))+(al[1]
            *C(t[1])))+(C(t[0])*(d[5]*S(t[3])*S(t[4])+d[1]));

    T[2][0]=-C(t[1]+t[2])*S(t[4])*C(t[5])-(S(t[1]+t[2])*((C(t[3])*C(t[4])
            *C(t[5]))-(S(t[3])*S(t[5]))));

    T[2][1]=C(t[1]+t[2])*S(t[4])*S(t[5])+(S(t[1]+t[2])*((C(t[3])*C(t[4])
            *S(t[5]))+(S(t[3])*C(t[5]))));

    T[2][2]=-(S(t[1]+t[2])*C(t[3])*S(t[4]))+(C(t[1]+t[2])*C(t[4]));

    T[2][3]=d[0]-(al[1]*S(t[1]))-(al[2]*S(t[1]+t[2]))+(d[3]*C(t[1]+t[2]))
            -(d[5]*((S(t[1]+t[2])*C(t[3])*S(t[4]))-(C(t[1]+t[2])*C(t[4]))));

    T[3][0]=0;
    T[3][1]=0;
    T[3][2]=0;
    T[3][3]=1;


    M1x=T[0][3]-(d[5]*T[0][2]);
    M1y=T[1][3]-(d[5]*T[1][2]);
    M1z=T[2][3]-(d[5]*T[2][2]);
    M1ox=T[0][3];
    M1oy=T[1][3];
    M1oz=T[2][3];

    if(iter==1)
    {

        /* le vecteur "ob" represente les coordonnees du point de depart */

    ob1=T[0][3]-(d[5]*T[0][2]);
    ob2=T[1][3]-(d[5]*T[1][2]);
    ob3=T[2][3]-(d[5]*T[2][2]);
    ob4=T[0][3];
    ob5=T[1][3];
    ob6=T[2][3];
    }

}
```

```
/*  ================================================================
    *              SOUS-PROGRAMME DE CALCUL DE LA POSITION          *
    *                                                               *
    *                    Par Josiane JUEGOUO. 1993.                 *
    ================================================================ */

#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)


extern int rep,m,iter;
extern int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
extern float *Tab,**Q,*t,*a,*al,*d,*Tt,*Ttmax,STEP,*pas_bal,stepR,stepT;
extern float *J[6],Dx[729],Dy[729],Dz[729],DOx[729],DOy[729],DOz[729];
extern float Epx,Epy,Epz,Eox,Eoy,Eoz,Erx,Ery,Erz,Erox,Eroy,Eroz;
extern float M1x,M1y,M1z,M1ox,M1oy,M1oz,R0,R1,Ri,T[4][4];
extern float M2x,M2y,M2z,M2ox,M2oy,M2oz,ex,ey,ez,eox,eoy,eoz;
extern int Ct,Cr,ind,ind1,ind2;
extern float Mx,My,Mz,Mox,Moy,Moz,eps1,eps2,ob1,ob2,ob3,ob4,ob5,ob6,ar,br,cr;
extern double ECR;
float m12,m22,m32,m42,m52,m62,test1,test2,Ar2,Br2,Dr2,ar2,br2,cr2;
extern int fact,test3;
float tem;

/* Calcul des elements de la matrice jacobienne de position */

jacobpos()
{


 J[0][0]=-S(t[0])*((d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2]))+(al[1]
         *C(t[1])))-(d[1]*C(t[0]));

 J[0][1]=-C(t[0])*((-d[3]*C(t[1]+t[2]))+(al[2]*S(t[1]+t[2]))
         +(al[1]*S(t[1])));

 J[0][2]=-C(t[0])*((-d[3]*C(t[1]+t[2]))+(al[2]*S(t[1]+t[2])));

 J[0][3]=0;
 J[0][4]=0;
 J[0][5]=0;

 J[1][0]=C(t[0])*((d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2]))+(al[1]
         *C(t[1])))-(S(t[0])*d[1]);

 J[1][1]=-S(t[0])*((-d[3]*C(t[1]+t[2]))+(al[2]*S(t[1]+t[2]))+(al[1]
         *S(t[1])));

 J[1][2]=-S(t[0])*((-d[3]*C(t[1]+t[2]))+(al[2]*S(t[1]+t[2])));
```

```
J[1][3]=0;
J[1][4]=0;
J[1][5]=0;
J[2][0]=0;

J[2][1]=-((d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2]))+(al[1]*C(t[1])));

J[2][2]=-((d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2])));

J[2][3]=0;
J[2][4]=0;
J[2][5]=0;


        Erx=fabs(M2x-M1x);
        Ery=fabs(M2y-M1y);
        Erz=fabs(M2z-M1z);

        Ar2=fabs(M2x-ob1);
        Br2=fabs(M2y-ob2);
        Dr2=fabs(M2z-ob3);
        ar2=fabs(M2ox-ob4);
        br2=fabs(M2oy-ob5);
        cr2=fabs(M2oz-ob6);

        /* R1 correspond a la distance entre la position actuelle du
           bras et la position visee du bras. Apres chaque iteration,
           R1 designe toujours la plus grande distance. */

        R1=sqrt(pow(Erx,2)+pow(Ery,2)+pow(Erz,2));

        R0=0;
        Ct=0;

        for (i=0; i<Nb_comb; i++)
        {


        Dx[i]=((J[0][0]*Q[i][0])+(J[0][1]*Q[i][1])+(J[0][2]*Q[i][2])
              +(J[0][3]*Q[i][3])+(J[0][4]*Q[i][4])+(J[0][5]*Q[i][5]));

        Dy[i]=((J[1][0]*Q[i][0])+(J[1][1]*Q[i][1])+(J[1][2]*Q[i][2])
              +(J[1][3]*Q[i][3])+(J[1][4]*Q[i][4])+(J[1][5]*Q[i][5]));

        Dz[i]=((J[2][0]*Q[i][0])+(J[2][1]*Q[i][1])+(J[2][2]*Q[i][2])
              +(J[2][3]*Q[i][3])+(J[2][4]*Q[i][4])+(J[2][5]*Q[i][5]));

       /* tem est une variable permettant de verifier si,
        dans la combinaison utilisee, une au moins des
        articulations d'orientation recevra une commande
        non nulle. */

       tem=(fabs(Q[i][3]))+(fabs(Q[i][4]))+(fabs(Q[i][5]));

        m12=M2x-(M1x+Dx[i]);
        m22=M2y-(M1y+Dy[i]);
        m32=M2z-(M1z+Dz[i]);
```

```
        /* Ri represente la distance entre la position actuelle augmentee
           d'un deplacement elementaire et la position visee. */

        Ri=sqrt(pow(m12,2)+pow(m22,2)+pow(m32,2));

        Epx=fabs(m12);
        Epy=fabs(m22);
        Epz=fabs(m32);

    if(tem==0)
    {
     if(Ri<R1)
       {

       Ct+=1;
       R1=Ri;
       Erx=Epx;
       Ery=Epy;
       Erz=Epz;
       ind1=i;

       }

    }
     else
       {

       if((Ri>0)&&(Epx<Erx)&&(Epy<Ery)&&(Epz<Erz))
       {

        R1=Ri;
        Erx=Epx;
        Ery=Epy;
        Erz=Epz;
        ind2=i;

        }

       }
       if((Ri==R1)&&(Ri<eps1))
       {
       ind=i;
       goto egal;
       }

}

    if(Ct>0)
    {
    ind=ind1;
    }
    else
    {
    ind=ind2;
    }
```

```
egal:    Mx=M1x+Dx[ind];
         My=M1y+Dy[ind];
         Mz=M1z+Dz[ind];

         ex=fabs(M2x-Mx);
         ey=fabs(M2y-My);
         ez=fabs(M2z-Mz);


 printf("\n iter= %d\n",iter);
 printf("\n Coordonnees du point initial:\n");
 printf("%g   %g   %g     %g   %g   %g\n",ob1,ob2,ob3,ob4,ob5,ob6);
 printf("\n Coordonnees du point actuel:\n");
 printf("%g   %g   %g     %g   %g   %g\n",M1x,M1y,M1z,M1ox,M1oy,M1oz);
 printf("\n Combinaison selectionnee : %d\n",ind);
 printf("\n%g  %g  %g  %g  %g  %g\n",Q[ind][0],Q[ind][1],Q[ind][2],Q[ind][3],Q[i
 printf(" Coordonnees du point transitoire:\n");
 printf("%g   %g   %g     %g   %g   %g\n",Mx,My,Mz,T[0][3],T[1][3],T[2][3]);
 printf("%g  %g  %g  \n",ex,ey,ez);
 printf("\n Coordonnees du point objectif:\n");
 printf("%g   %g   %g     %g   %g   %g\n",M2x,M2y,M2z,M2ox,M2oy,M2oz);
 }

  int signe(float x)
  {
  if(x>=0)
  {
      return(1);
  }
  else
  {
  return(0);
  }
  }
```

```
/*  ======================================================================
    *            SOUS-PROGRAMME DE CALCUL DE L'ORIENTATION               *
    *                                                                    *
    *                    Par Josiane JUEGOUO. 1993.                      *
    ====================================================================== */

#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <alloc.h>
#define Calloc(m,type) (type *)calloc((m),sizeof(type))
#define TRUC float
#define C(x) cos(x)
#define S(x) sin(x)


extern int rep,m,iter;
extern int i,n,Nb_comb,j,k,s,A,B,N,Nb_trans,*Trans;
extern float *Tab,**Q,*t,*a,*al,*d,*Tt,*Ttmax,STEP,*pas_bal;
extern float *J[6],Dx[729],Dy[729],Dz[729],DOx[729],DOy[729],DOz[729];
extern float Epx,Epy,Epz,Eox,Eoy,Eoz,Erx,Ery,Erz,Erox,Eroy,Eroz;
extern float M1x,M1y,M1z,M1ox,M1oy,M1oz,R0,R1,Ri,T[4][4];
extern float M2x,M2y,M2z,M2ox,M2oy,M2oz,ex,ey,ez,eox,eoy,eoz;
extern int Ct,Cr,ind,ind1,ind2;
extern float Mx,My,Mz,Mox,Moy,Moz,eps1,eps2,ob1,ob2,ob3,ob4,ob5,ob6,ar,br,cr;
extern double ECR,u1,v1,w1,u2,v2,w2,ui,vi,wi,THETA,Tet1,Teti;
extern double fonc(double,double,double,double,double,double);
 float m13,m23,m33,m43,m53,m63,Ar3,Br3,Dr3,ar3,br3,cr3;
extern int fact,test3;
float tp[6],eps3=0.0035;

/* Le calcul de L'orientation n'utilise pas de matrice jacobienne,
   mais plutot la fonction sinusoidale definie dans le programme
   principal. La matrice jacobienne de position permet neanmoins
   de s'assurer que la position calculee ne change pas.*/

jacobori()
{

 u1=T[0][2];
 v1=T[1][2];
 w1=T[2][2];

 Tet1=THETA;                    /*fonc(u2,u1,v2,v1,w2,w1);*/

 J[0][0]=-S(t[0])*((d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2]))+(al[1]
         *C(t[1])))-(d[1]*C(t[0]));

 J[0][1]=-C(t[0])*((-d[3]*C(t[1]+t[2]))+(al[2]*S(t[1]+t[2]))+(al[1]*S(t[1])));

 J[0][2]=-C(t[0])*((-d[3]*C(t[1]+t[2]))+(al[2]*S(t[1]+t[2])));

 J[0][3]=0;
 J[0][4]=0;
 J[0][5]=0;
```

```
J[1][0]=C(t[0])*((d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2]))+(al[1]
        *C(t[1])))-(S(t[0])*d[1]);

J[1][1]=-S(t[0])*((-d[3]*C(t[1]+t[2]))+(al[2]*S(t[1]+t[2]))+(al[1]*S(t[1])));

J[1][2]=-S(t[0])*((-d[3]*C(t[1]+t[2]))+(al[2]*S(t[1]+t[2])));

J[1][3]=0;
J[1][4]=0;
J[1][5]=0;
J[2][0]=0;

J[2][1]=-((d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2]))+(al[1]*C(t[1])));

J[2][2]=-((d[3]*S(t[1]+t[2]))+(al[2]*C(t[1]+t[2])));

J[2][3]=0;
J[2][4]=0;
J[2][5]=0;


        for (i=0; i<Nb_comb; i++)
        {

        Dx[i]=((J[0][0]*Q[i][0])+(J[0][1]*Q[i][1])+(J[0][2]*Q[i][2])
               +(J[0][3]*Q[i][3])+(J[0][4]*Q[i][4])+(J[0][5]*Q[i][5]));

        Dy[i]=((J[1][0]*Q[i][0])+(J[1][1]*Q[i][1])+(J[1][2]*Q[i][2])
               +(J[1][3]*Q[i][3])+(J[1][4]*Q[i][4])+(J[1][5]*Q[i][5]));

        Dz[i]=((J[2][0]*Q[i][0])+(J[2][1]*Q[i][1])+(J[2][2]*Q[i][2])
               +(J[2][3]*Q[i][3])+(J[2][4]*Q[i][4])+(J[2][5]*Q[i][5]));


        Ri=sqrt(pow(Dx[i],2)+pow(Dy[i],2)+pow(Dz[i],2));

        fact=fabs(Q[i][0])+fabs(Q[i][1])+fabs(Q[i][2]);

     if(Ri==0)
     {
     for (j=0; j<n; j++)
       {
       tp[j]=t[j]+Q[i][j];
       }

    T[0][2]=C(tp[0])*((C(tp[1]+tp[2])*C(tp[3])*S(tp[4]))+(S(tp[1]+tp[2])*C(tp[4]

    T[1][2]=S(tp[0])*((C(tp[1]+tp[2])*C(tp[3])*S(tp[4]))+(S(tp[1]+tp[2])*C(tp[4]

    T[2][2]=-(S(tp[1]+tp[2])*C(tp[3])*S(tp[4]))+(C(tp[1]+tp[2])*C(tp[4]));


    ui=T[0][2];

    vi=T[1][2];

    wi=T[2][2];
```

```
    Teti=fonc(u2,ui,v2,vi,w2,wi);
    if((fabs(Teti)<fabs(Tet1))&&((Q[i][3]!=0)||(Q[i][4]!=0)||(Q[i][5]!=0)))
    {
    Tet1=Teti;
    ar3=ui;
    br3=vi;
    cr3=wi;
    ind=i;

    }


   }

}
       THETA=fonc(u2,ar3,v2,br3,w2,cr3);
       m13=Mx+(d[5]*ar3);
       m23=My+(d[5]*br3);
       m33=Mz+(d[5]*cr3);


printf("\n\n Coordonnees du point actuel:\n");
printf("%g   %g   %g     %g   %g   %g\n",ob1,ob2,ob3,ob4,ob5,ob6);
printf("\n\n Coordonnees du point vise:\n");
printf("%g   %g   %g     %g   %g   %g\n",M2x,M2y,M2z,M2ox,M2oy,M2oz);
printf("\n\n Coordonnees du point final:\n");
printf("%g   %g   %g     %g   %g   %g\n",M1x,M1y,M1z,m13,m23,m33);
printf("\n\n Nombre d'iterations : %d     Combinaison selectionnee : %d\n",iter,

}
```